# KNOWLEDGE TRANSFER IN ROBOT MANIPULATION TASKS

A Dissertation
Presented to
The Academic Faculty

by

Jacob O. Huckaby

In Partial Fulfillment
of the Requirements for the Degree
Doctor of Philosophy in
Robotics

School of Electrical and Computer Engineering
Georgia Institute of Technology
May 2014

# KNOWLEDGE TRANSFER IN ROBOT MANIPULATION TASKS

Approved by:

Professor Henrik I. Christensen,
Advisor
School of Interactive Computing
*Georgia Institute of Technology*

Professor Aaron Bobick
School of Interactive Computing
*Georgia Institute of Technology*

Professor Ayanna Howard
School of Electrical and Computer
Engineering
*Georgia Institute of Technology*

Professor Andrea Thomaz
School of Interactive Computing
*Georgia Institute of Technology*

Professor Michael Beetz
Institute for Artificial Intelligence
*University of Bremen*

Date Approved: 4 April 2014

*To Emily,*

*Who stuck with me through rain and shine.*

# ACKNOWLEDGEMENTS

A great many people have contributed to this work in various ways. First I should thank my advisor, Henrik Christensen. When I first came to Georgia Tech as a recently accepted Robotics PhD student I had no money and no prospects. Henrik had the heart to take me in as a graduate research assistant, and gave me the chance to work with some great robots. Henrik, ever the quintessential robotics renaissance man, helped open my eyes to the wider world of robotics research, problems, and applications. I am grateful for the many conversations and lessons learned, and how he helped teach not just the importance of good research, but grounding that research in application. I can say that I am a better researcher, student, and mentor from following Henrik's example.

I would also like to thank my committee members, Aaron Bobick, Michael Beetz, Ayanna Howard, and Andrea Thomaz for their willingness to serve on my thesis committee and their patience and feedback during the dissertation process. Their opinions, thoughts, and constructive comments were invaluable in helping me improve my research.

I would like to thank Boeing for funding the majority of my graduate work. I have been able to work on a number of interesting projects, and learn a great deal more about manufacturing than I ever thought I would know.

Our work on planning coupled with the knowledge transfer framework would have been much more difficult without our collaboration with Professor Stavros Vassos at Sapienza University of Rome. I am grateful for his time, help, and collaboration.

The members of the Cognitive Robotics Lab have been excellent labmates and friends, and have helped me a great deal through the course of my graduate career. I

would also like to thank the members of the Robotics Graduate Student Organization.

I should also thank several people I had the opportunity to work with while I was at BYU, without whom I might not have discovered the amazing field of robotics. Dr. Doran Wilde was a great mentor and extremely helpful in getting my feet under me with respect to initial work in robotics and research in general. Thanks also to Dr. James Archibald and Dr. D.J. Lee. I also would not have succeeded in my first endeavor in robotics without the help and advice of Beau Tippetts, Spencer Fowers, and Kirt Lillywhite.

Last I would like to thank my family. My mom and dad for their constant love and support, my grandparents for their love and kind words. My children, Isabelle, Noah, Seth, and Gus for sacrificing and being patient when daddy had to work instead of reading bedtime stories. And last but most certainly not least, my wonderful wife Emily, who sacrificed and worked and gave up so much for us to be able to succeed in graduate school. Her patience and love have made it all possible.

# TABLE OF CONTENTS

# LIST OF FIGURES

# SUMMARY

Technology today has progressed to the point that the true potential of robotics is beginning to be realized. However, programming robots to be robust across varied environments and objectives, in a way that is accessible and intuitive to most users, is still a difficult task. There remain a number of unmet needs. For example, many existing solutions today are proprietary, which makes widespread adoption of a single solution difficult to achieve. Also, most approaches are highly targeted to a specific implementation. But it is not clear that these approaches will generalize to a wider range of problems and applications. To address these issues, we define the Interaction Space, or the space created by the interaction between robots and humans. This space is used to classify relevant existing work, and to conceptualize these unmet needs. GTax, a knowledge transfer framework, is presented as a solution that is able to span the Interaction Space. The framework is based on SysML, a standard used in many different systems, which provides a formalized representation and verification. Through this work, we demonstrate that by generalizing across the Interaction Space, we can simplify robot programming and enable knowledge transfer between processes, systems and application domains.

# CHAPTER I

# INTRODUCTION

## *1.1 Motivation*

Robotics has tremendous potential. For decades people of all walks of life have envisioned the multitude of different ways robots could make a positive benefit to society - from domestic service robots to help with household-oriented tasks, to manufacturing robots that work collaboratively with human workers to complete complex industrial objectives. Technology today has progressed to the point that we are beginning to see this potential realized to varying degrees. Examples include Simon [9], a research platform for studying human-robot interaction, and the Baxter platform [53], a product for the manufacturing domain. A recent product of Rethink Robotics, Baxter is a humanoid robot with two 7-DOF arms that is designed to work with humans in collaborative efforts in manufacturing environments. It is outfitted with sensors and software to allow users to teach the robot tasks to perform. These platforms are an important step forward, toward a true shared effort between human laborers and robots to achieve complex manufacturing objectives, which has long been a holy grail of robotics in industrial manufacturing.

Yet despite these many advancements in the state of the art, programming robots to be robust across varied environments and objectives, and in a way that is accessible and intuitive to most users, is still a difficult task, and there remain many unmet needs with the existing technology. For example, while Baxter has the capability of being programmed by demonstration to perform specific tasks, if a parameter of the task changes (such as updating to a different type of screw), the entire task will need to retaught to the robot for it to be able to successfully complete the updated task.

None of the other knowledge from the previous case can be saved or reused.

This problem of dynamically changing task requirements is common in manufacturing, and the problem with brittle solutions is not limited to just this one platform. This is true for most applications involving industrial robots, and is a significant hurdle for the widespread adoption of robotics and automation technologies in many different manufacturing industries. In a study carried out by the European Commission regarding the outlook of the robotics industry in Europe [19], an analysis was given of the robot value chain. It stated that in the cost breakdown of implementing and deploying a robotic solution to a problem in industry, the actual robot consists of only 25% of the full cost, where the other 75% comes from integration costs. The highest single cost (at 40%) is in fact the programming of the robot. And as manufacturing needs require that these robots must be used in solving a variety of different problems, while some cost can be saved in reusing the robot itself, the 75% implementation cost must be paid again and again for each new problem. Thus in this context, the biggest hurdle for robotics is in fact not the robot itself, but rather is in its deployment in specific applications.

The heart of this issue can be framed as a knowledge transfer problem. In both cases there is an inability to represent the knowledge in the system in such a way that lends itself to modularity, and the ability to be reused in a different context or problem. By representing the knowledge such that it is portable and reusable in different scenarios, this could reduce the programming cost of redeployment in different applications, potentially saving up to 40% of the total cost of the system. It is important to note that there are different types of knowledge transfer that can be referred to. These include human-robot knowledge transfer, when some type of knowledge is shared between human and robot (e.g. such as that seen in Programming by Demonstration, which is almost exclusively human-robot knowledge transfer), and robot-robot knowledge transfer, or when knowledge is shared between robots or systems with no human

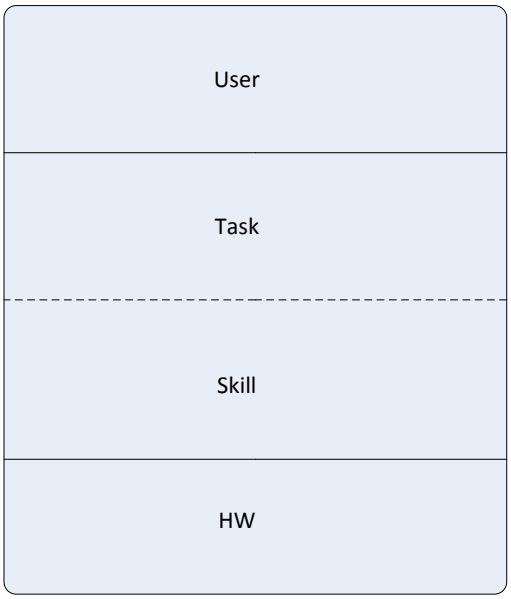
intervention. To achieve this level of knowledge transfer, a method is needed that will span the Interaction Space.

## *1.2 Technical Approach*

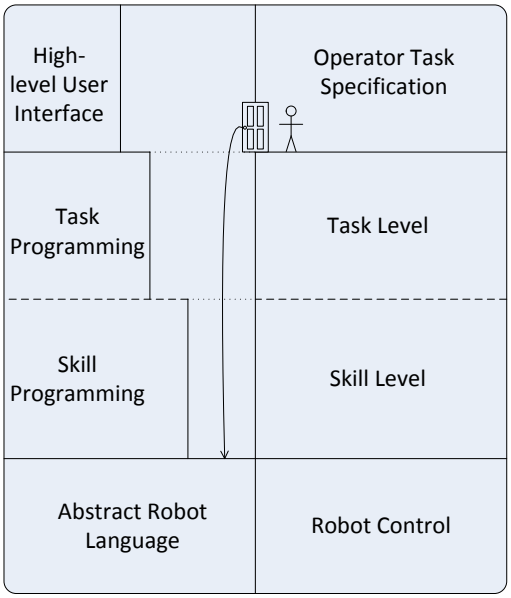
### 1.2.1 The Interaction Space

The ***Interaction Space*** is defined as the space created by the intersection of robots operating in human environments, and human-robot collaboration. Figure 1a shows the Interaction Space divided into different areas. At the lowest level is the robot level, or direct robot control. At the highest level is the user level. The remaining space has been divided into two different areas. The skill level represents low-level abstractions above the robot level. The task level represents higher-level abstractions and concepts. In the context of the Interaction Space, the different types of knowledge transfer can be seen quite easily. Knowledge can be transferred vertically from human to robot, making the robot accessible to the human and human knowledge open to the robot. Knowledge can also be shared horizontally on the graph between different robot platforms and processes, making existing robot knowledge open and usable across different contexts and applications. Thus it can be seen that the problems of effective knowledge transfer and spanning this Interaction Space created by human-robot collaboration are one and the same.

Figure 1b shows different levels of programming interaction (left) paired with the respective level of specification (right). At the lowest level is that of the abstract robot language, or the individual robot scripting language for creating motion and action in the real world. Much of what happens in robotics today happens at this level. Yet if we imagine the user being at the top level, the level of intuitive interaction with the system, it is a considerable distance down to this level where most interaction occurs, and this transition can be a difficult hurdle for many users. As we ascend, we see more abstract levels of interaction, namely those of skill-level interaction (such as




(a)




(b)

Figure 1: The Interaction Space.

discrete-event systems) and task-level interaction (such as programming by demonstration). The top consists of the high-level, intuitive user interface. There has been much work (including the previous Baxter example) that has attempted to create a connection between intuitive interaction and low-level control, making it straight forward for an operator to program the system to perform a task. However, as a general solution these attempts ultimately fail due to the fact that the implementations do not generalize across the interaction space, and updates to the problem specification require a different solution.

### 1.2.2 GTax: A SysML-based Knowledge Transfer Framework

To address these issues, in this thesis we propose the use of SysML as the language for the creation of a framework used for modeling robot capabilities in a given application domain. SysML has a number of advantages as a choice for modeling language, including that it is an industry standard for modeling complex systems and as such has a formal specification that is well studied. The framework itself takes the form of an abstraction hierarchy, where the lowest level of abstraction is represented as hardware independent atomic actions, called skill primitives, that a system would be required to perform to accomplish tasks in the application domain. This abstraction hierarchy allows users and designers to dynamically select an appropriate level of abstraction for system- and task-specific needs.

Using this generalized representation makes it possible to apply these skills across a broad range of problems, from a multitude of different tasks and task descriptions, to tasks across a variety of different platforms and systems. Using the representation presented in the framework keeps the abstraction level high enough to generalize across many different problem descriptions, but also across different platforms, as the skill primitives exist above the hardware implementation level. The representation also brings the specification of these tasks up to a more conceptually straight-forward

level, making it easier for a user to interface with, and to interact with the robot, without having to take time to learn a lower-level description of individual robot capabilities and parameter specifications. Through the work performed in this dissertation, we will show how framework can be used to span the interaction space, and thereby enable knowledge transfer in a robot application domain.

## *1.3 Dissertation Overview*

### 1.3.1 Thesis Statement

Knowledge transfer across different contexts, and high level interaction with robot manipulation systems, can be enabled through the use of a knowledge transfer framework that can effectively span the interaction space.

### 1.3.2 Contributions

In this research we present GTax, a knowledge transfer framework that will span the interaction space, and make it possible for end users to intuitively interact with the system, enabling knowledge transfer across different contexts. This is done by using a hierarchical task decomposition model that uses existing industry-standard modeling tools to capture representational requirements and elements of intuitive interaction. Specifically, GTax addresses the thesis of this work by making the following contributions:

- The creation of a taxonomy-based knowledge transfer framework that is able to span the interaction space, by representing knowledge in such a way as to be able to generalize across different processes and hardware platforms.
- A method to generate complete objective execution plans from user-defined high-level task descriptions.

# CHAPTER II

# RELATED WORK

In this chapter we present related research efforts that have addressed aspects of knowledge transfer and the interaction space (figure 2). These include efforts on knowledge representation, learning by demonstration, low-level systems, and knowledge transfer frameworks.

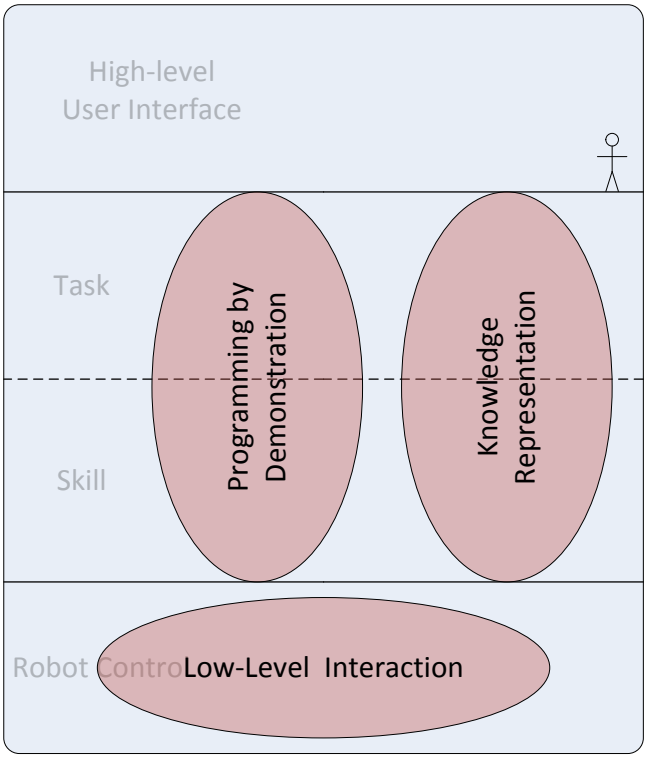



Figure 2: Coverage of the interaction space.

Discussion will begin with a brief look at the low-level interactions with the hardware. Next will be the important role of knowledge representations, and how they have been utilized in different ways. After that will be PbD approaches and how these various methods have been applied. Last we will introduce several different types of

knowledge transfer frameworks.

## *2.1 Low-Level Interaction*

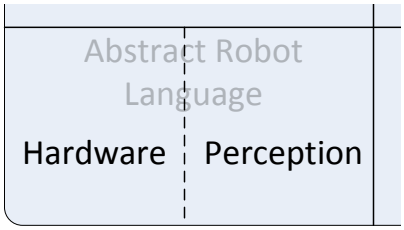



Figure 3: The lowest level can be divided into two sections: hardware and perception.

Referring to the interaction space, the lowest level can be divided into two parts (figure 3). The first section represents the robot itself, while the second section represents perception for the robot, such as computer vision. At this low level, there exists certain abstraction models, or systems that make these low-level systems easier to use. For the hardware, examples of this include the Robot Operating System, or ROS [51], the OROCOS system [64], and the JAUS standard [57]. ROS and similar packages operate as a Robot Abstraction Model (RAM), or as a way to provide a common interface to the hardware itself. For perception, examples include the Open Computer Vision package (OpenCV [5]) and the Point Cloud Library (PCL [58]). These type of packages act as a Perception Abstraction Model (PAM), or a layer above specific sensors to perform useful perception functions. These abstraction models are popular and useful tools for a trained robot technician, but themselves are still fairly low-level. Despite this, many real-world robot implementations exist at this low level. The problem is that this lowest common denominator is too low for the kind of high-level interaction needed for non-expert operators, especially when looking to go across different contexts such as platforms and processes. Here, we want a common interface, where this interface needs to abstract away some of these often difficult technical details that can crop up with the low-level abstraction models.

In addition to these abstraction models, a number of languages have been created to make the programming of robots and tasks easier. These include the abstract, robot-specific languages such as KRL for KUKA robots and Karel for Fanuc robots. For general platforms, languages such as the Task Description Language (TDL [61]) and Execution Support Language (ESL [22]) were created.

## *2.2 Knowledge Representation*

Given that programming robots is a difficult task requiring considerable technical knowledge, a great deal of prior work has focused on addressing how to make robot programming more intuitive and accessible to operators with less technical experience. Much of this work has focused on finding efficient and effective methods for representing system knowledge to accomplish this task, with respect to certain requirements and constraints on the system. Many different types of knowledge representations have been proposed.

### 2.2.1 Skill Level

One philosophy has worked on encoding task knowledge as a function of motion. Examples of this type of representation include dynamical systems [30] and object-action complexes [37]. The work of Lyons, et. al. [42] defined a model for robot computation using port automata. Kosecka, et. al. [35] used a discrete event systems framework to model tasks and behaviors for robotics. More recent work includes that of Dantam, which takes a grammar-based approach to represent sensorimotor information [12].

### 2.2.2 Task Level

Another philosophy has espoused the view that one can effectively represent important system knowledge symbolically, such as using skill trees [34] or topological task graphs [1]. This symbolic approach to knowledge representation assumes that the

system has more inherent knowledge (it knows how the relationships between the symbols and physical instantiations behave), while it allows for the modeling of more high-level concepts than motion-based representations. Work by Kress-Gazit, et. al., [36], [18] uses linear temporal logic to model task specifications to produce correct robot controllers for different tasks. Some researchers have used this symbolic approach to address the issue of knowledge reuse or knowledge transfer in areas outside of manufacturing, including [15], [16], [44].

There is also work that has been done in representing specifically manufacturing and assembly objectives, such as in the application of Petri Nets [55]. Another approach is the work of de Mello and Sanderson [27], which uses AND/OR graphs to enumerate all possible paths through the assembly process to get to the overall objective (e.g. an assembled product.) The paper then proposes the use of a graph search algorithm to find an appropriate path through the graph based on specific problem specifications.

Ontologies have been used in many different frameworks as a representational tool for sharing knowledge (primarily semantic or relational knowledge) between systems. Systems like ConceptNet [41] (which utilizes the MIT Open Mind Common Sense database [62]) and CYC [39] attempt to model human commonsense knowledge as a relational ontology, with a small, predefined set of semantic relationships determining how concepts are tied together. KNOWROB [65] is a robot-specific framework that bases its knowledge representation on an ontological implementation, here using OWL2 [26] as the ontology standard. The ontology is defined around concepts and relations important for robots operating in the real world. Work by Lim et al. has proposed a similar ontology-based approach for robot knowledge in indoor environments [40].

Recent work has also been done by Balakirsky, et al. [2] to define a knowledge representation for industrial robotics. This representation is similarly based on the

OWL ontology standard. In this work the authors attempt to provide a method of structuring knowledge in such as way as to be able to reuse it for different problems. This work also proposes a multi-layered knowledge representation. Our work likewise proposes a multi-layered representation, but at a different level of abstraction. It also differs fundamentally from the work of Balakirsky in that this representation is proposed to improve not only modularity in knowledge, but also usability and intuitiveness for users.

## *2.3 Programming by Demonstration*

Programming by Demonstration (PbD)[21], [33], [13], [54], [45], also referred to as Learning by Demonstration or Imitation Learning, is a research area whose underlying goal is human-robot knowledge transfer. In this approach, a user or operator is tasked with teaching a robotic system how to perform some task or objective by interacting with the robot in some way. This interaction can take on several different forms. Often the operator will physically demonstrate the task being performed, and the system will use some type of visual perception to obtain information from the user, such as cameras or 3D sensors to capture the motion of the user and the relevant features in the environment. Processing is then done on this information to relate the physical movement of the operator to changes in the environment, and subsequently relate this information to operations that must be performed to accomplish the task. In this way, relevant knowledge related to the performance of the task is transferred into the system. Other methods for interacting with the system include physical interaction, either by moving the robot end effector by hand to show the task, or by controlling the robot using another type of interface (e.g. joystick, graphical interface, etc.) Similar information is obtained by monitoring the robot's own performance, and inferences are made to generalize knowledge to relate this to task completion.

Within PbD there are many different approaches and methods suggested to program a robot through human interaction. These different approaches fit in different places in the interaction space shown in figure 2. It should be noted though that while some specific PbD work is clearly situated in low-level skill or high-level task acquisition, much work is more difficult to categorize, both in differences in definitions between skills and tasks, as well as efforts by various researchers to do some of both of what is defined in this work as skills and tasks. However, while these methods may be able to capture more of the interaction space vertically, it is not clear that these methods generalize horizontally across the space.

### 2.3.1 Skill Level

Some work fits clearly in the skill interaction level. Peters [49] uses motion primitives to categorize human motion and task execution. The work of Calinon [7] uses machine learning techniques such as Gaussian Mixture Models to categorize skills in terms of different motions. Skubic focuses on using force data to learn low-level assembly skills [63]. Other examples include Ude [67] and Hersch [25].

### 2.3.2 Task Level

Other work fits better at the task level in the interaction space. Examples include work such as that of Kuniyoshi, who extracted task descriptions by watching humans perform assembly tasks [38], Dillman, who has used human activity analysis to learn task descriptions for household tasks [14], and Cakmak, whose work involves using keyframes to understand task specifications [6]. Work has been done in task learning by Pardowitz [47] and Nicolescu [44], and PbD has also been applied to the problem of task modeling by Ekvall [16].

### 2.3.3 Programming by Instruction

Another method of transferring human knowledge into a robotic system is Learning by Instruction. This approach relies on utilizing a previously specified language for communicating with the system about various concepts that have been defined in the language. Many PbD systems, for example, will use natural language processing to allow users to interact with the system in a natural, intuitive way. Here the language for communicating with the robot is actual human spoken language, although the system is often configured to understand only a small subset of the spoken language limited to a few simple concepts. Other systems have been implemented that use human language in the form of text. For example, Tenorth et al. [66] used written instructions to inform the robot about how to complete some high-level task. While this approach is similar to using spoken dialog, it employs a different method for using natural language to interact with the system. Still others have used different modalities for system interaction to give instruction, such as graphical interfaces [15].

## *2.4 Knowledge Transfer Systems*

The last research area presented in this section is that of actual systems or frameworks that are specifically designed for knowledge transfer. Given a specific knowledge representation, a number of other projects have addressed the problem of representing, storing, and transferring knowledge between various robotic systems.

### 2.4.1 SIARAS

In the SIARAS (Skill-based Inspection and Assembly of Reconfigurable Automation Systems) project [60], a system is developed to assist in the automatic reconfiguration of automation systems. This is done due to the need for light-weight (low overhead) processes to address current manufacturing demands. System components are designed both to represent skills and parameters, as well as the process flow. This is

done using a specific ontology. A skill server is designed to aid a human operator to match process requirements with the representations in the database.

### 2.4.2 ROSETTA

Another project working on robot deployment in manufacturing environments in the ROSETTA (RObot control for Skilled ExecuTion of Tasks in natural interaction with humans; based on Autonomy, cumulative knowledge and learning) project [56]. This project tries to design industrial robotic systems that are suitable for working around and collaborating with humans in the manufacturing process. One important aspect of their approach is a skill repository, where skills can be shared across systems. Machine learning methods are applied to interactions with humans to improve those skills over time.

### 2.4.3 RoboEarth

RoboEarth [68], [69] is a project aimed at creating a global repository for all knowledge relevant to robotic systems, not just action-specific information, but everything a robot would need to function in a dynamic, real-world environment, including relevant environmental information, maps, object models, and semantic information. The architecture is organized in three layers, with the top layer being the global database, which acts as an information server, a second layer containing hardware independent functions such as action recognition and semantic mapping, and a bottom layer consisting of robot-specific implementations. Knowledge representation and processing is handled by the KNOWROB system [65].

# CHAPTER III

# A KNOWLEDGE TRANSFER FRAMEWORK FOR ROBOTICS

## *3.1 Introduction*

The state of robotics today is advancing at an impressive rate. The need for robotics and automation solutions continues to increase across many different industries, from manufacturing and field robotics to healthcare and home service applications. And the solutions proposed for these needs continue to improve as well. As robotics comes to take a greater part in these various industries and segments, and the needs and interactions become more varied and complex, the robotics and automation systems themselves become more varied and complex. As this complexity grows, so to does the need to produce methods for accurately modeling these systems. Without appropriate models for the systems, it will become increasingly difficult to characterize these systems and ensure that they are reliable and safe. This need goes hand in hand with the need to improve the standardization of these practices in the field of robotics. The field of robotics and autonomous systems still has much to accomplish in terms of standardization, though there have been efforts to move forward on the formulation of standards. In this case as well, a great number of benefits can be gained by standardizing the modeling of these complex autonomous systems as much as possible.

In this chapter we propose the use of the Systems Modeling Language (SysML) as a viable language for modeling autonomous systems and robotics. As a descriptive graphical modeling language it has a number of features that make it appealing to use on complex systems. It is also a standardized modeling language with a thorough formal specification, potentially making it easier to communicate and share

information regarding these systems, and to create benchmarks and compare systems to one another.

## *3.2 Systems Modeling Language (SysML)*

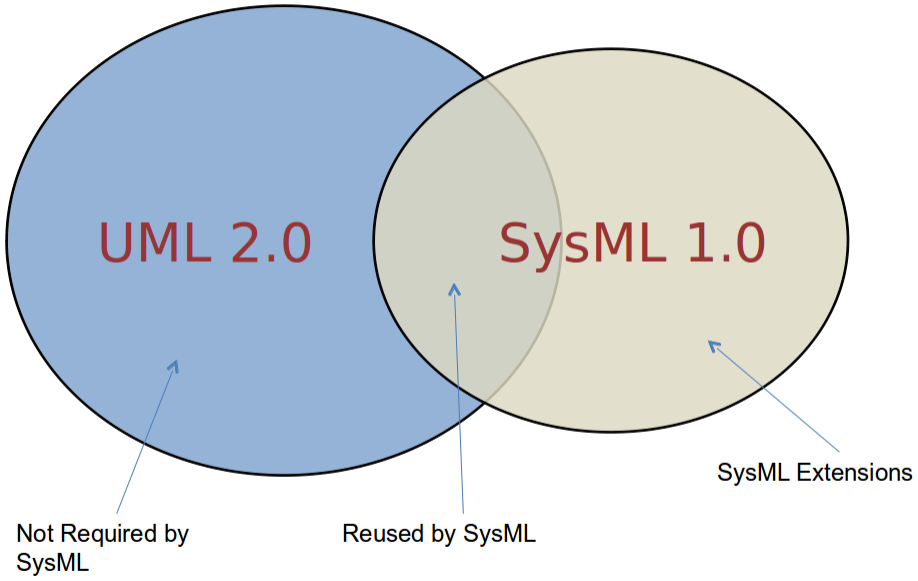



Figure 4: The relationship between UML and SysML.

The Systems Modeling Language, or SysML [46], is a standardized graphical modeling language designed for modeling processes in a systems engineering context. Specifically, it was designed for the specification, analysis, and verification of complex physical systems [24].

SysML is based on the popular UML (Unified Modeling Language) standard. UML is a generalized modeling language designed specifically for software engineering, and while it was applied to systems engineering for a number of years, there were a number of difficulties in modeling real systems with the software-specific UML. A solution was proposed in the form of SysML, which was presented not so much as a new language for describing systems, but as an extension or modification of the

existing UML specification. Figure 4 illustrates this relationship.

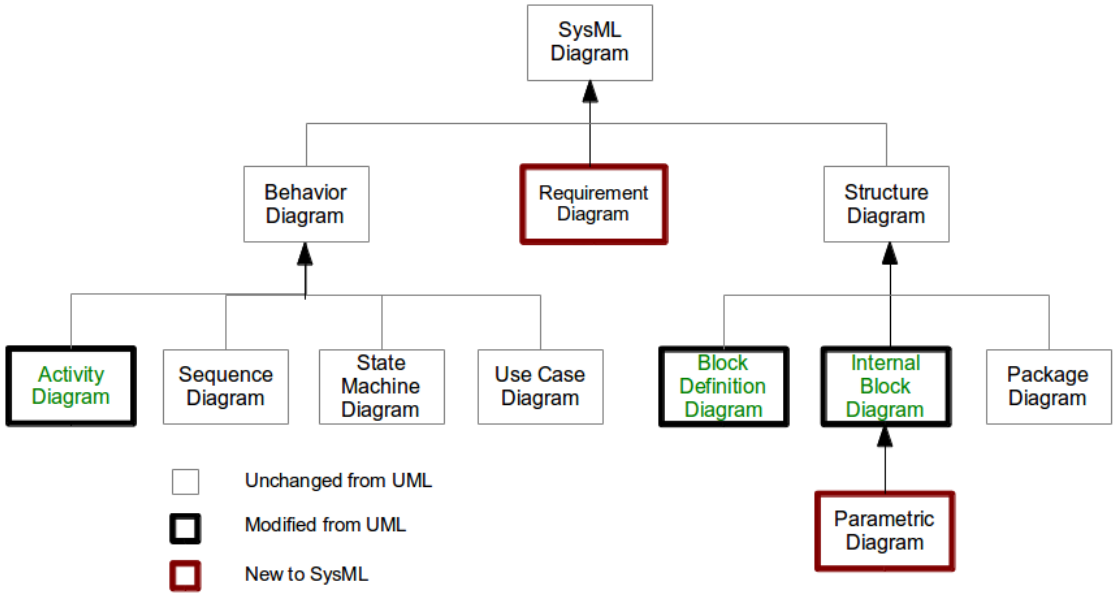



Figure 5: Diagram relationships between UML and SysML.

SysML borrows a great deal of its structure from UML. It is composed of a collection of different diagrams (which are used for modeling different aspects of the system.) While UML contains thirteen different modeling diagrams, SysML contains only nine. These relationships are shown in figure 5. Specifically, of the nine different possible diagrams used in modeling systems with SysML, only two are unique to SysML (the requirement diagram, and the parametric diagram.) The others are either modified from the UML diagrams, or carried over directly from the UML specification.

The nine modeling diagrams available in the SysML Specification can be grouped into four different categories, commonly known as the four pillars of SysML: Structure Diagrams, Behavior Diagrams, Requirement Diagrams, and Parametric Diagrams. A brief description of these categories and the modeling diagrams they represent is given below.

### 3.2.1 Structure Diagrams

The basic structural unit is SysML is the **block**. The structure diagrams describe the static relationships between blocks in the system. The three structure diagrams in SysML are the Block Definition Diagram, the Internal Block Diagram, and the Package Diagram.

#### *3.2.1.1 Block Definition Diagram*

The Block Definition Diagram (BDD) is a core diagram in SysML. It defines the structural hierarchy of a system as well as system components. It is similar to the UML Class Diagram.

#### *3.2.1.2 Internal Block Diagram*

The Internal Block Diagram (IBD) describes the internal structure of a given block in the system. It is a redefinition of the Composite Structure Diagram from UML.

#### *3.2.1.3 Package Diagram*

The Package Diagram is used to organize the model, and defines the model structure. It remains unchanged from the UML specification.

### 3.2.2 Behavior Diagrams

The behavior diagrams describe the dynamic relationships between blocks in the system. The four behavior diagrams are Use Case Diagrams, Sequence Diagrams, Activity Diagrams, and State Machine Diagrams.

#### *3.2.2.1 Use Case Diagrams*

The Use Case Diagram provides a high-level description of the interaction among systems. It is unchanged from the UML specification.

#### *3.2.2.2 Sequence Diagrams*

The Sequence Diagram describes the interactions between parts of a system. It is unchanged from the UML specification.

#### *3.2.2.3 Activity Diagrams*

The Activity Diagram describes the flow of data and control between activities.

#### *3.2.2.4 State Machine Diagrams*

The State Machine Diagram describes the state transitions and actions that a system performs in response to events. It is unchanged from the UML specification.

### 3.2.3 Requirements Diagram

The Requirements Diagram describes the system requirements, requirements hierarchies and relationships that satisfy or verify the requirements.

### 3.2.4 Parametric Diagram

The Parametric Diagram describes quantitative constraints on system properties such as performance, reliability, and mass properties, and serves as a means to integrate the specification and design models with engineering analysis models.

## *3.3 SysML Related Work*

A number of different methods have been proposed for modeling robotics systems over the years. Petri nets have been a popular choice due to their mathematical properties [55]. Lyons, et. al [42] proposed using port automata to model the robot as a means to develop a useful robot programming language. Kosecka [35] proposed the use of a finite state automata framework for modeling discrete event systems for autonomous systems. Dantam, et. al [12] has defined a new formal grammar known as the motion grammar as a means for modeling the system and its capabilities. What these approaches share in common is that the underlying representations for each of

them are compatible with formal methods, which allow them to be used to make specific statements about some properties of the systems they model (feasibility, etc.) Unfortunately these methods also have several drawbacks. It is not clear that they all scale well to highly complex systems. Some of these approaches are also not able to fully model many robot systems, as many systems deal not only with the robot but also environmental and human interactions. To address some of these issues, some work has been aimed at attempting to combine these modeling approaches with descriptive models.

SysML, on the other hand, models systems through the use of descriptive graphical diagrams that capture system structure and behavior from a variety of perspectives. It is a formal language in that the descriptions are fully specified, though it does not share the same properties of the other modeling languages mentioned previously. It does however have some advantages, including being able to capture a wide range of system abstractions. SysML has been utilized for modeling a number of different applications across many different industries. Popular examples range from water purification [20] to satellite system design [59].

While SysML has not yet gained wide spread adoption or popularity in robotics, there have been several groups that have attempted to integrate the SysML modeling language into the design of their systems. Examples include that of Chhaniyara, et al. [10]. In this work the authors present SysML applied to the modeling of complex space robotic systems used in satellite servicing missions. They argue that SysML is a good option because of the automated requirement verification and model tracing that is possible with SysML, which can be used to reduce cost and time in the system engineering. Rahman, et al. [52] makes a similar case for using SysML in the design of mobile robots. In this work the software for a mobile robot is developed using the model-based system engineering (MBSE) approach. The claim is that using MBSE (with SysML as the specific implementation) can enable the creation of reusable

software modules for programming the robot to allow platform independent design and reduced development time. While this work demonstrates that SysML can be applied to some robotic systems with varying degrees of success, they do not describe how it might be applicable to other robot application domains.

## *3.4 Discussion*

There are a number of important considerations to take into account when looking at selecting an appropriate modeling language for systems such as robotics.

In the introduction we pointed out two important things in the further development of modeling in robotics: the ability to accurately model increasingly complex systems, and the standardization of such an approach. SysML is uniquely suited to both of these tasks. As a general purpose graphical modeling language designed for systems in particular, a developer is able to select the level of abstraction and thus the apparent complexity of the system in the model design itself. This ability to select the appropriate level of abstraction depending on the problem or system being dealt with is important, while it also makes it easier to communicate about the system to others. This can be important especially when dealing with large systems, as stakeholders often need to communicate regarding disparate system components.

The use of SysML would also be a step toward model standardization in robotics. When UML was initially introduced to software developers it quickly developed into the standard for modeling software design project. Likewise SysML is becoming increasingly popular as a means for describing system models.

### 3.4.1 Verification and Validation

One charge that has been leveled at the use of SysML in robotics is that it is in some respects inferior to approaches that can apply formal methods to the verification of certain system properties. To properly address this claim, we first define verification and validation in the right context. In the context of systems engineering, verification

and validation (V&V) is typically taken to mean the following: [4]

- Verification - to establish the truth of the correspondence between a system and its specification
- Validation - to establish the fitness of a system for its intended purpose

Informally, verification is thought of as "Is one building the system right?" while validation asks "Is one building the right system?" These questions are fundamentally linked to system specification and requirements. Formal verification, on the other hand, is concerned with proving the correctness of some property using formal methods. As far as requirements verification and validation, SysML is able to facilitate this due to the language specifications and given a correct requirements diagram. It should be noted here also that is often the software tool that is supporting SysML modeling that determines what features are available than the language itself. In fact, many packages offer tools for requirements verification and validation along with simulation and code generation, among others. Formal verification is another matter. While the SysML representation does not inherently support formal methods for verification, there has been a great deal of work to work around this limitation, while still utilizing the strengths of SysML. For example, work has been done to provide probabilistic verification of SysML models [31]. Work has also been done to transform SysML models into other representations for the purpose of formal verification [8].

### 3.4.2 Automatic Code Generation

Automatic Code Generation is another important feature that is available with many software tools that support modeling with SysML. Automatic code generation can take several different forms. Many tools support forward engineering or reverse engineering the model. In forward engineering (in this context) code is generated based on the SysML model that is defined through the various diagrams used. Reverse

engineering the code generates (or more often updates) a model based on changes that take place in the code. Another version of automatic code generation is known as round-trip engineering. In round-trip engineering, the system allows for changes to made both to the model and to the the code base, and updates the code or the model based on what changes were made.

The ability to monitor and update the model in this way has great implications for robotics and large scale systems in general. One large potential source of errors could come from neglecting to keep changes updated and propagated throughout a system with a very large code base and a significant number of models (or a single large, highly complex model.) Propagating changes by hand throughout a complex system is time consuming and error prone. This can help keep system models up to date an accurate for validation or simulation of system tasks.

## *3.5 A SysML-Based Framework for Knowledge Transfer in Robotics*

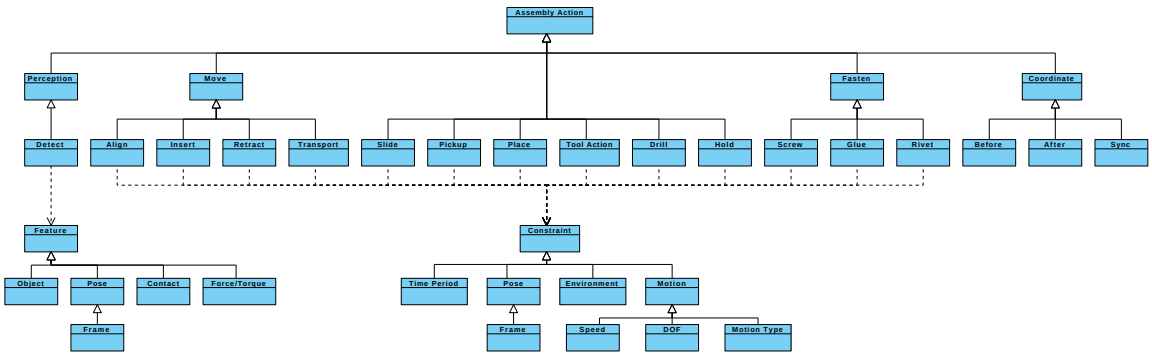



Figure 6: A taxonomy of capabilities for the assembly domain.

In order to effectively model robot manipulation tasks in a given domain is it important to be able to capture a wide range of capabilities for the system. To that end we define an application domain dependent taxonomy for describing robot manipulation tasks. In order to demonstrate how this would work for a specific use case, we will use the manufacturing assembly domain as an example domain for

describing some of the advantages of this approach. Figure 6 shows a taxonomy defined for the robot manufacturing assembly domain. One underlying assumption is that the taxonomy is defined in such a way that it is able to model all possible tasks and objectives in the domain. This is an important foundation for the claim that the framework is able to generalize across all tasks in the given domain. In this case, the taxonomy is defined using a SysML block definition diagram for specifying these system capabilities.

### 3.5.1 Skill Primitives, Skills, & Tasks

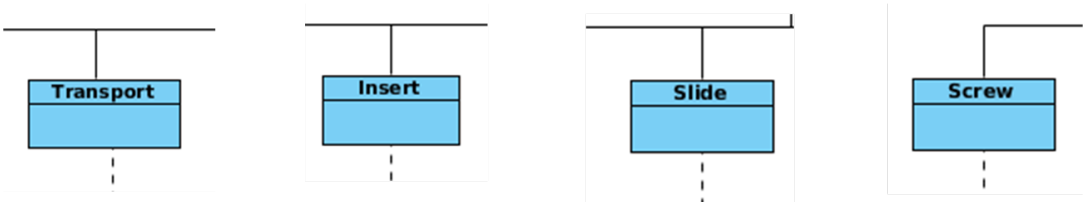



Figure 7: Examples of different skill primitives.

The taxonomy itself is composed of several different components. The top half of the taxonomy in figure 6 consists of a set of what are termed *skill primitives*. Skill primitives are defined as the basic, atomic actions that a robot is able to perform in the domain, and represent a generalized capability. Examples include the ***Transport*** skill primitive for moving the robot through the environment, as well as the ***Screw*** skill primitive for performing screwing tasks with the end effector (see figure 7.)

In the terminology of the framework, a skill then is defined an abstraction of a skill primitives. One example is the ***Fasten*** skill, which can be thought of as an abstraction of several different types of skill primitives in the assembly domain (see figure 8.)

*Tasks* refer to high-level concepts, that are defined as a collections of skills. An example of this would be assembling a wheel assembly for the vehicle model.

The overall goal or objective then can be defined in terms of a sequence of tasks.

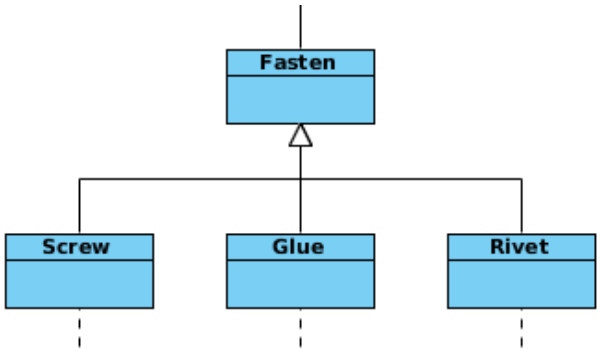



Figure 8: The ***Fasten*** skill is divided into several different skill primitives; ***Screw***, ***Glue***, and ***Rivet***.

This kind of hierarchy fits naturally into the interaction space, as seen in figure 9.

### 3.5.2 Constraints

The lower half of the taxonomy represents the parameters or constraints used to specify the skill primitives (figure 10.) The skill primitives are defined in terms of the parameters placed on them by the system requirements for how the action is to be executed. Having a way to specify these requirements and parameters is essential to being able to accurately model the system.

For example, ***Time Period*** can be used by the ***Hold*** skill primitive to describe how long something should be held in a specified pose, as well as indicating that some action must be completed in a certain amount of time. Actions such as ***Align*** and ***Transport*** will often be done with respect to some goal ***Pose***. Examples of ***Environment*** constraints include workspace limitations and restrictions, the need to follow a particular surface, and constraints in which motion planning through the environment is done in the presence of obstacles. ***Motion*** constraints relate to the action motion of the robot, and include ***Speed***, ***DOF***, and ***Motion Type***, which can be used to specify how a robot would need to follow a trajectory with a specific velocity profile, such as in a welding task.

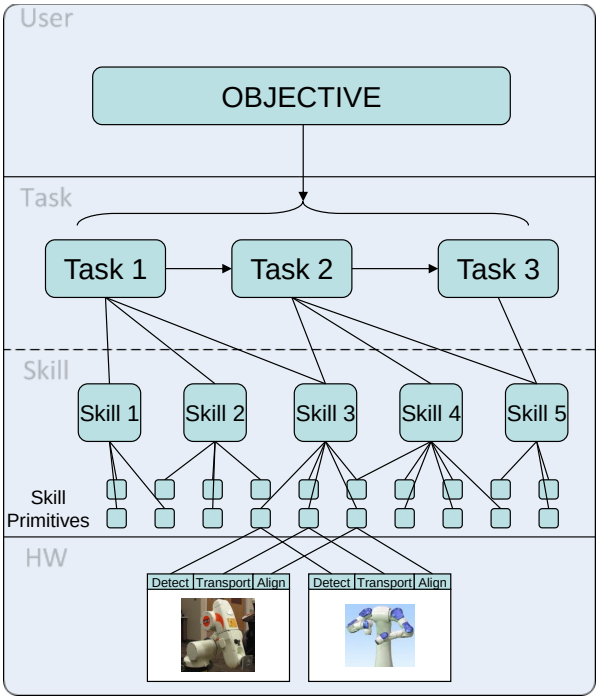



Figure 9: Hierarchy of task decomposition. Consider how a robot interacts with its environment. Robots interact with the world by working toward some objective, such as constructing an airplane wing. These objectives can be decomposed into a sequence of tasks that must be accomplished to successfully achieve the objective, such as attaching two sheets of metal together using a bolt. Tasks can further be broken into skills required to achieve the task, like threading a bolt, and what are called in this work skill primitives, which are simple, robot-specific actions like closing a gripper around an object.

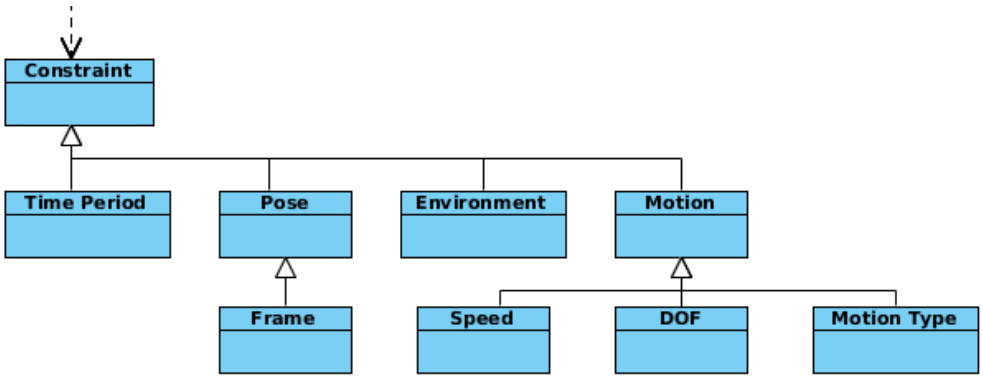



Figure 10: Parameters for skill primitives.

### 3.5.3 Perception

A collection of required robot capabilities must include all actions the robot needs to complete the task, including perceptual capabilities. This type of representation for perceptual actions is one feature that is missing from many of the systems mentioned in the section on related work.

Many different types of perception are required to execute an assembly task, and the taxonomy is able to facilitate this interaction. The ***Detect*** skill primitive (figure 11) is essentially the interface module into these different perception actions. Constraints or parameters on this skill primitive are defined as features to be detected, and include ***Object***, ***Pose***, ***Contact***, and ***Force/Torque***.

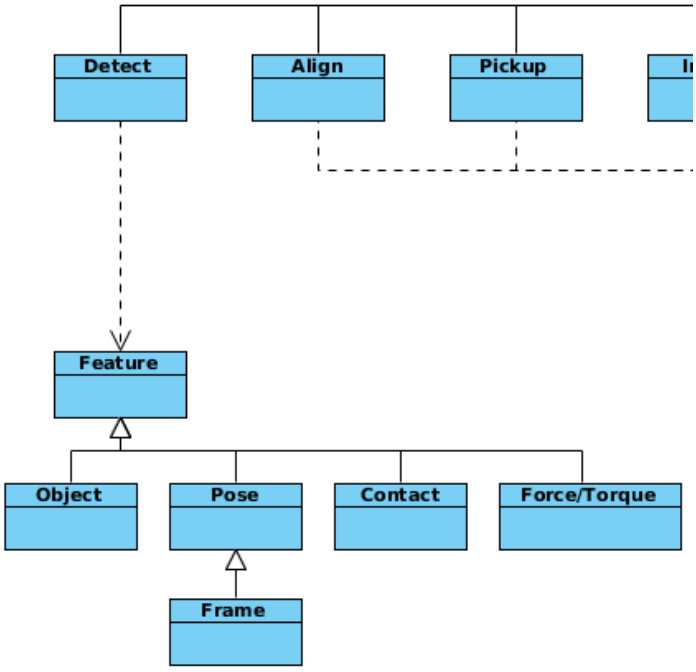



Figure 11: ***Detect*** skill primitive.

These feature types handle the various types of perception that is required in assembly. This would cover sensors and algorithms that have to do with pose estimation and object detection, such as cameras and 3D scanners, and other methods of finding and localizing desired objects for manipulation. Other sensors of interest that

are useful in this context include contact sensors and force/torque sensors for detecting contact within the environment and performing complex, accurate manipulation tasks.

### 3.5.4 Control

Equally as important to the performance of the assembly task as the interaction with the perception subsystem is the ability of the taxonomy to interact with the control subsystem. With the exception of ***Detect*** and ***Coordinate***, all skill primitives interact with the control subsystem by calling on some controller to perform some action. ***Insert*** will call on a controller to change the position of the end effector, ***Slide*** will control on the contact interactions at the tool tip to slide the workpiece along some surface. There are many different types of controllers that are needed by the system to do assembly.

It should be mentioned here that the taxonomy is designed to be independent of implementation. Just as ***Transport*** does not depend on the path planning algorithm used to plan the path to be taken, neither does it depend on the type of controller utilized to drive that path.

The control subsystem and the perception subsystem are tightly integrated. In most cases, the controllers called into action by the skill primitives will in turn call perception functions to provide feedback to the system. This is true for closed loop control as well as open loop control, as the pose specification is needed to initialize even the open loop case. In the closed loop case, for example, ***Align*** will depend on the perception subsystem for the robot's pose to identify when it has achieved its goal. The perception system will often provide terminal conditions for the controller, which will in turn signal the system that the task is over and a new task can be attempted.

### 3.5.5 Other Features

#### *3.5.5.1 Coordination*

One essential capability to be able to be flexible and robust enough to represent all types of manufacturing assembly tasks is the ability to encapsulate the coordination of multiple systems operating in concert to achieve the manufacturing objective. Ideally the representation would be able to handle robots of differing configurations and specifications (e.g. both robots with identical specifications tasked with different roles, such as identical robots on opposite sides of a conveyor line, as well as robots with different specifications entirely, such as serial manipulators working with parallel platforms or mobile platforms.)

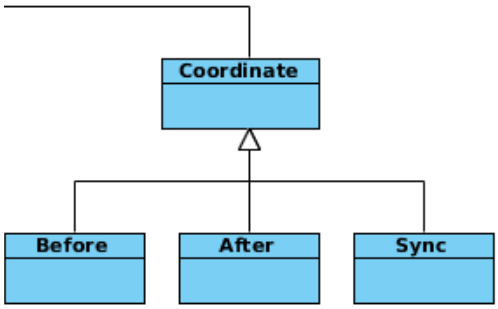



Figure 12: Coordinate skill

#### *3.5.5.2 Tool Use*

Another important aspect to consider with robots working in manufacturing environments is making sure that the representation can accommodate the ability to use tools in the accomplishment of the desired task. Tools are required to complete most complex manufacturing tasks. This important requirement is handled in our framework in two ways. First, the constraint parameterization allows for specifications to be given in different frames of reference, indicated by the ***Frame*** constraint. The reference frame associated with a particular tool could then be given whenever the

tool is in use by the robot.

An essential function needed by the robot in order to use a tool is the ability to actuate (or otherwise activate and use) the tool. For this reason the ***Tool Action*** skill primitive exists in the taxonomy. This enables the system to functionally operate the tool, such as activating a drill, or actuating the mechanism for dispensing glue on a gluing tool.

#### *3.5.5.3 Mobility*

For the proposed framework to capture the capabilities of various robotic systems, the concept of mobility must also be addressed. In the context of the assembly task, there are several different types of robots that may use this skill, from simple mobile platforms to full mobile manipulation systems.

The taxonomy that has been discussed in the previous sections can also be used to model mobility, by utilizing the ***Transport*** and ***Hold*** skill primitives. By using a different parameterization, ***Transport*** can be used to move a mobile platform from one configuration to another, just as it can move the end effector (or some object in the end effector) of a manipulator to a desired location. Similarly ***Hold*** can be used in the same context as a manipulator, instructing the robot (in this case a mobile platform) to hold a specific configuration for a certain amount of time.

## *3.6 Comparison to Standard Benchmarks*

In this section we examine how the proposed framework stands up against standard benchmarks. One commonly used benchmark in the manufacturing assembly domain that we will examine is the Cranfield Benchmark.

### 3.6.1 Cranfield Benchmark for Assembly

One method used to verify and compare proposed solutions to a given set of problems is through the use of benchmarks. One such benchmark, called the Cranfield

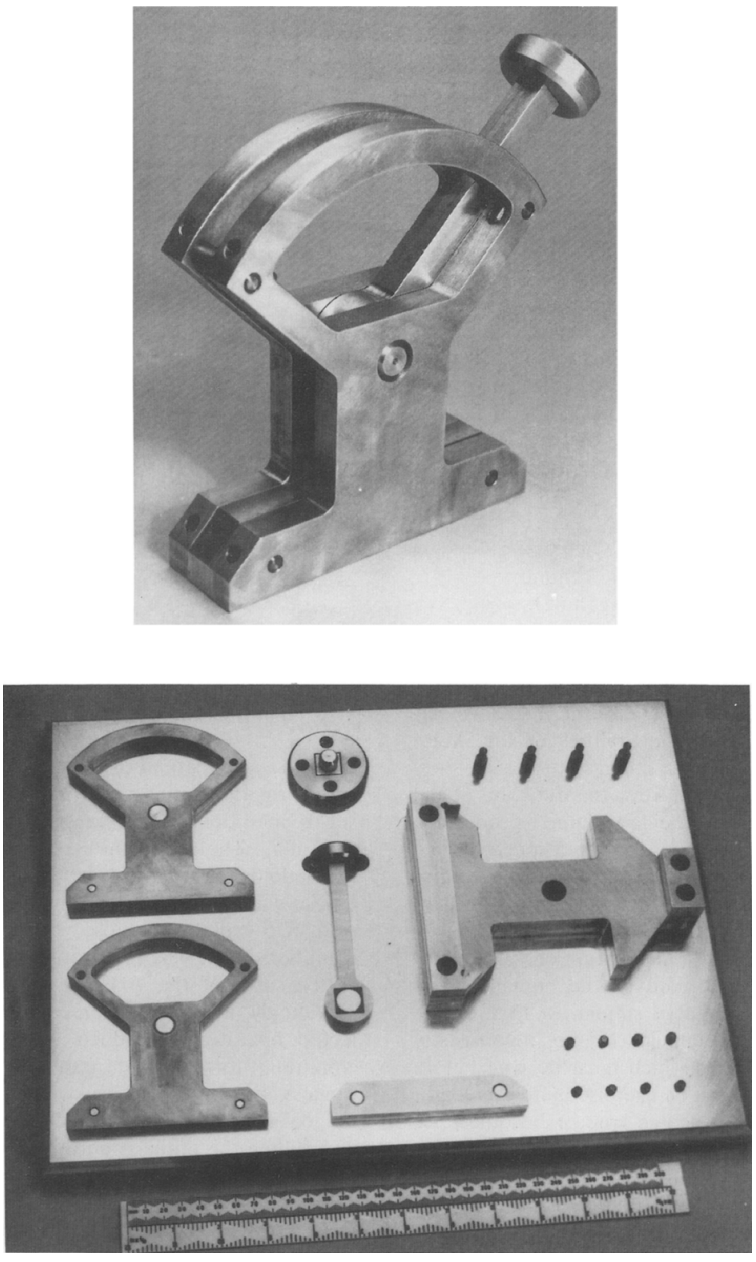


Figure 13: Cranfield Benchmark [50].

Benchmark [11], was developed by the Cranfield Institute of Technology to serve as a European benchmark for robot programming software and systems proposed to solve robot assembly tasks in manufacturing settings. It was devised as a canonical scenario that requires many of the typical assembly subtasks a robot is required to perform and most frequently occurring assembly problems encountered during an assembly operation, such as pick, place, and peg-in-hole (or insert) type operations. Using this benchmark one can compare many features of given robot control software solutions, including representation, control speed, motion type, etc. The SysML based method for describing robot skills in manufacturing settings presented here is able to successfully model the assembly task set forth in the Cranfield Benchmark.

The Cranfield Benchmark reduces the required set of actions a robot is required to perform to three: the ***Pick***, ***Place***, and ***Insert*** (or Peg-In-Hole) actions. These actions can be described in terms of a combination of simple motions with operations to open and close the gripper. As such, we are able to model the actions required to perform the benchmark using the task modeling framework. Figure 14 shows that we are able to model the benchmark task itself with a SysML sequence diagram.

Passing the Cranfield Benchmark demonstrates that the task modeling framework proposed here is at least sufficient for modeling basic assembly tasks. This successfully puts it on equal footing with many of the task modeling methods mentioned in the Related Work section. The next question is what the proposed framework offers that is unique to the other methods reviewed. We suggest that the representation allows us to dynamically select an appropriate level of abstraction to suit the specific task being modeled and used on a robot. These kinds of abstractions allow us to generalize in a way that many of these other methods do not allow. The way that we can generalize to allow skill abstraction across tasks, and task abstraction across different robot platforms, will be explored in the next chapter.

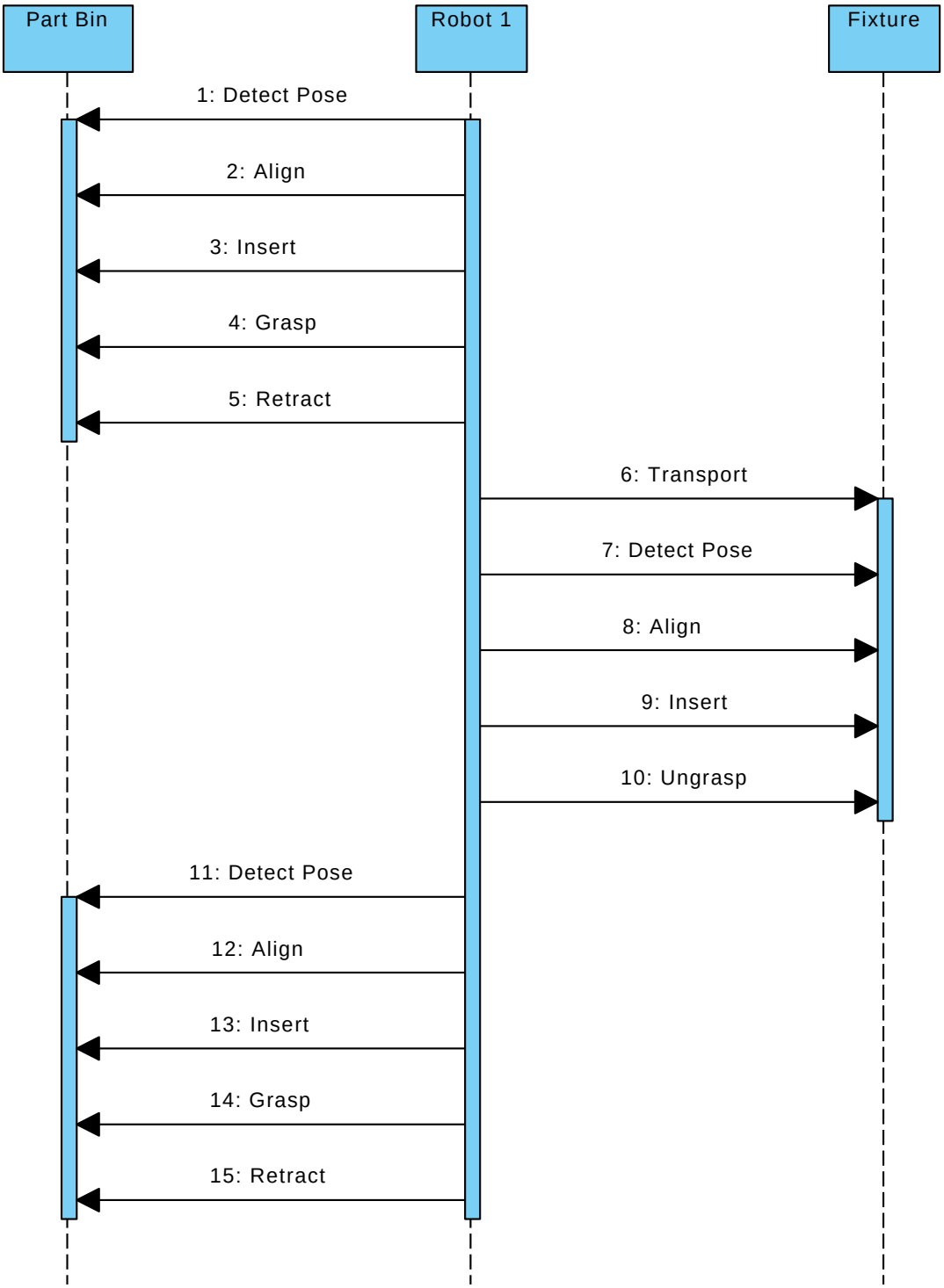



Figure 14: Sequence diagram for Cranfield Benchmark.

# CHAPTER IV

# SKILL ABSTRACTION

Often knowledge that exists in robot systems is represented in such a way that it cannot be reused, and much effort goes into reiterating this knowledge for new systems or new configurations of existing systems. This problem is especially true in the industrial manufacturing domain. In this chapter we present a task modeling framework based on SysML to aid in the making of knowledge persistent across processes and systems through the abstraction of robot skills. We present the framework and discuss its potential benefits. We also describe several experiments that have been performed to demonstrate the effectiveness of the approach to generalize models across different tasks and platforms.

## *4.1 Introduction*

Robotics has tremendous potential. For decades people of all walks of life have envisioned the multitude of different ways robots could make a positive benefit to society - from domestic service robots to help with household-oriented tasks, to manufacturing robots that work collaboratively with human workers to complete complex industrial objectives.

Yet despite the many advancements in the state of the art, programming robots to be robust across varied environments and objectives, and in a way that is accessible and intuitive to most users, is still a difficult task, and there remain many unmet needs with the existing technology.

This is especially true in manufacturing, where objective requirements and manufacturing process often change. This in turn requires a change in the system configuration and programming, which is time consuming and costly. What is needed is a

flexible solution that would allow robot users to easily reprogram and redeploy these robots, irrespective of changes made in the manufacturing process, or even the robot platform performing that process.

### 4.1.1 Problem Statement

Here we address the problem of inflexibility in current manufacturing processes that involve robotics and robot programming, and propose a solution in which our SysML-based task modeling framework is used to model the robot capabilities in a given application domain. By using this framework to model robot skills at a higher level of abstraction, we intend to show that we can introduce more flexibility into the manufacturing process by addressing robot programming and interaction at the task level.

## *4.2 Technical Approach*

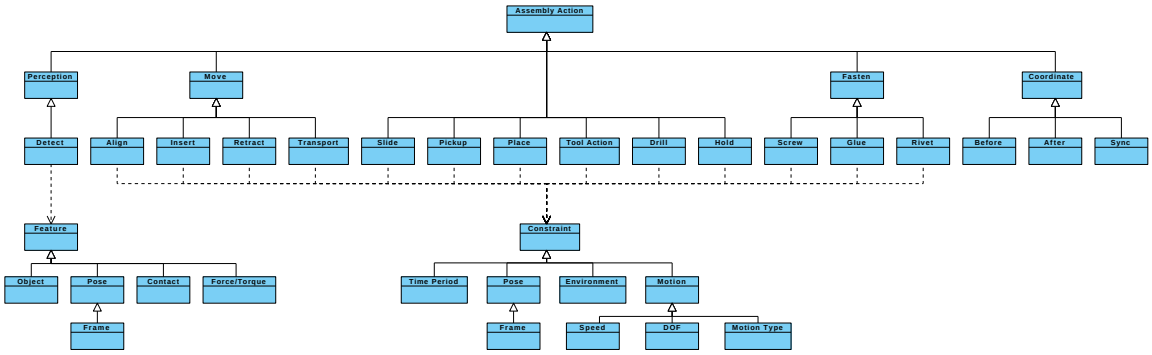
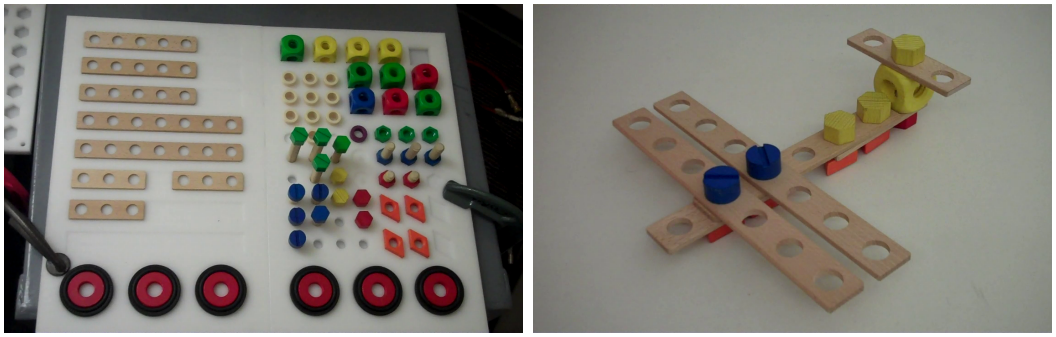



Figure 15: A taxonomy of capabilities for the assembly domain.

In order to effectively model robot manipulation tasks in a given domain it is important to be able to capture a wide range of capabilities for the system. To that end we define an application domain dependent taxonomy for describing robot manipulation tasks. For the sake of clarity we will use the manufacturing assembly domain as an example for describing some of the advantages of this approach. Figure 15 shows a taxonomy defined for the robot manufacturing assembly domain, which



(a) Assembly parts (b) Model Airplane

Figure 16: Example Assembly Task

was discussed in detail in the previous chapter. The taxonomy is defined such that it is able to model all possible tasks and objectives in the domain.

We propose that using a generalized high-level representation such as this makes it possible to apply these skill primitives to broad range of problems, and makes it possible to describe plans and task objectives in a way that is platform agnostic. In the following sections we describe a set of experiments which demonstrate that by using this task modeling framework, identical skill primitives can be used in varying combinations to perform a wide variety of tasks, and that identical task descriptions can be used to perform the same task on multiple different robot platforms.

### 4.2.1 A Motivating Example

This section goes into detail about why the example tasks are a good demonstration of the GTax framework. In this discussion, the objective is to be able to assemble the airplane model as seen in the right side of figure 16, using the component parts shown in the image on the left.

The question posed here considers what core capabilities are required for the robot to be able to perform this simple task. To determine this, we trace through each step involved in assembling the model.

To begin with, before anything can be done with the parts themselves, some means must be devised for the robot to be able to ***Detect*** where the parts are in the

environment, and distinguish between the different components to find the first piece that must be placed into the fixture for assembly. Once the correct part is found, the robot must then determine the ***Pose*** of the part, so that it can be manipulated. With the part identified, and pose estimation complete, the robot must move itself into a position such that it can pickup the part, so that it can ***Transport*** it to the correct place in the fixture. The robot then ***Aligns*** with the part, so that it can be grasped. At this point, the robot needs to pickup the piece; or, ***Open the Tool*** used to grasp the part, ***Insert*** the tool into position such that it can be grasped, ***Close the Tool*** around the part to the appropriate ***Grasp Width***, and ***Retract*** the tool back to the original position.

To determine where the part goes now, the robot again needs to ***Detect*** the location on the fixture, and do ***Pose Estimation*** to determine the correct placement in the fixture for the part. With the part in hand, the robot can ***Transport*** it to the appropriate fixture location. Here again, the robot will ***Align*** with the placement pose, then it will ***Insert*** the part into position, ***Open the Tool*** to release the part, and ***Retract*** to the starting position. The first part is now in place. This process is repeated for the remaining parts that are inserted directly into the fixture.

The next pieces follow a similar pattern, except that instead of ***Detecting the Pose*** and ***Aligning*** to the fixture, this is done with respect to a piece on the fixture. For example, for part B5 (a 3-board), this is done with respect to the orange nuts previously placed in the fixture. This continues for the next 5 pieces.

When these are placed, the next part needs to be screwed into the assemblage on the fixture. The part is retrieved in the same manner as the other parts. Once this is accomplished, the robot must ***Detect*** the place location and the ***Axis*** along which to insert the screw. The robot can then ***Transport*** the screw to the final location, ***Align*** to the screw axis, and ***Insert*** the screw while performing a ***Screw*** motion. To determine if the screwing is finished, the robot ***Detects*** the ***Torque*** of the end

effector, above some ***Threshold***. This process is repeated on the remaining parts until the model is finally assembled.

It can be seen from this example that only a few core capabilities (***Detect***, ***Transport***, ***Operate Tool***, ***Align***, ***Insert***, ***Retract***), and the parameters needed to specify those capabilities (***Pose***, ***Grasp Width***, ***Alignment Axis***, ***Torque***, ***Torque Threshold***), are required to complete the task.

## *4.3 Task Abstraction*

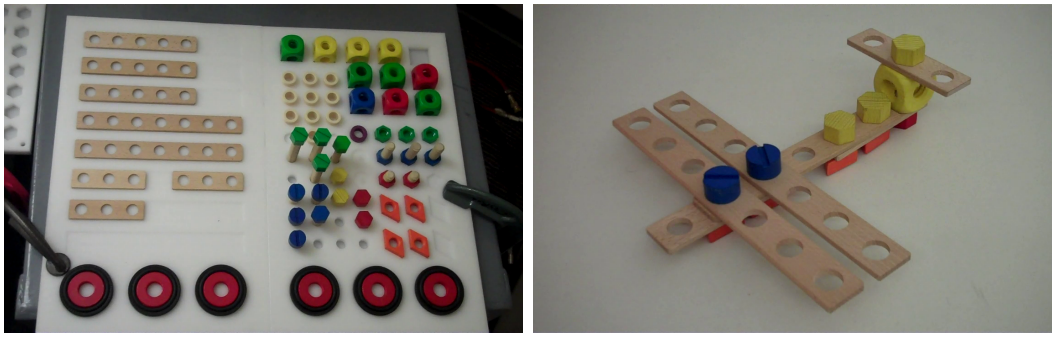
In this section we describe the experiments designed to demonstrate two different aspects of abstraction using the task modeling framework previously presented. The first experiment is intended to demonstrate abstraction across tasks, or that a single robot using the same skill primitives modeled in the framework is able to successfully perform two different tasks.

### 4.3.1 Experimental Design

In this experiment we will use the same robot to perform two different robot assembly tasks. For both tasks the robot will be responsible for assembling a toy model using small wooden model parts. These parts, seen in figure 17a include wooden screws, washers, nuts, and structural blocks.

#### *4.3.1.1 Airplane Assembly*

The first model selected was the airplane model seen in figure 17b. As discussed in the previous section, this was chosen because the task required the use of many of the skill primitives specified in the Assembly Taxonomy model, and assembly could be physically accomplished by a robot in a straight-forward manner. The parts used to assemble the plane are from a Baufix model kit, and are simple, brightly colored wooden construction parts such as screws, washers, blocks, nuts and boards (see figure 17a.) The model airplane was built using 17 individual parts. The assembly of



(a) Assembly Parts (b) Model Airplane

Figure 17: Model Airplane Assembly Task Used to Demonstrate Task Abstraction

the model airplane is done using a single manipulator and a static fixture designed specifically to aid the robot in this airplane assembly task.

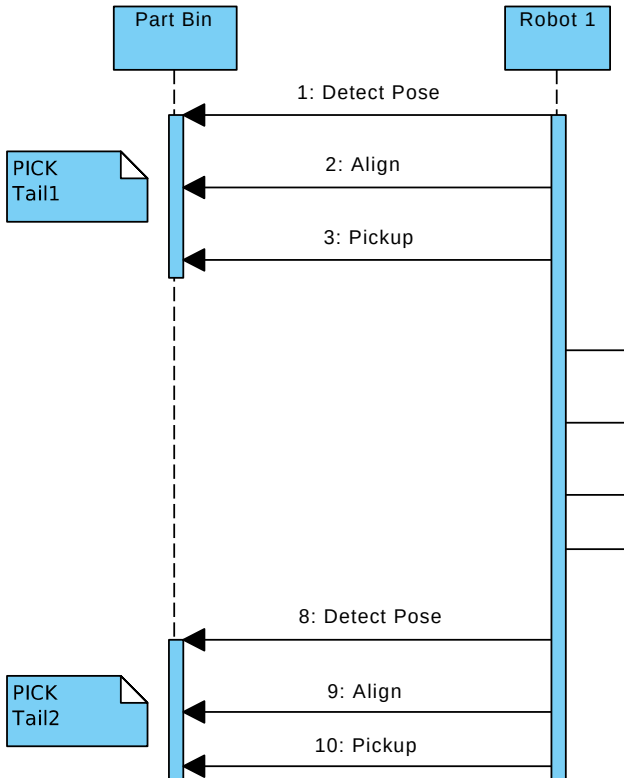



Figure 18: Portion of the sequence diagram for model airplane assembly.

The tasks themselves are modeled using a SysML sequence diagram. The sequence diagram is intended to show the organization of an assembly objective into a sequence of actions that are to be performed, and utilizes the skill primitives described in the

taxonomy as discrete steps in the sequence. An example of one of these sequence diagrams can be seen in figure 18. Each message from the robot to either the fixture or part bin represent an instantiation of a skill primitive (message 1 is a ***Detect***, message 2 is ***Align***, etc.) Despite the fact that the taxonomy is designed for the manufacturing assembly domain, the skill primitives themselves are the same and are simply used on toy parts instead of their more traditional manufacturing counterparts for demonstration purposes only. Note also that while only a portion of the sequence diagram is shown, the complete diagram consists of 258 actions, or instantiations of skill primitives, to complete the assembly task.

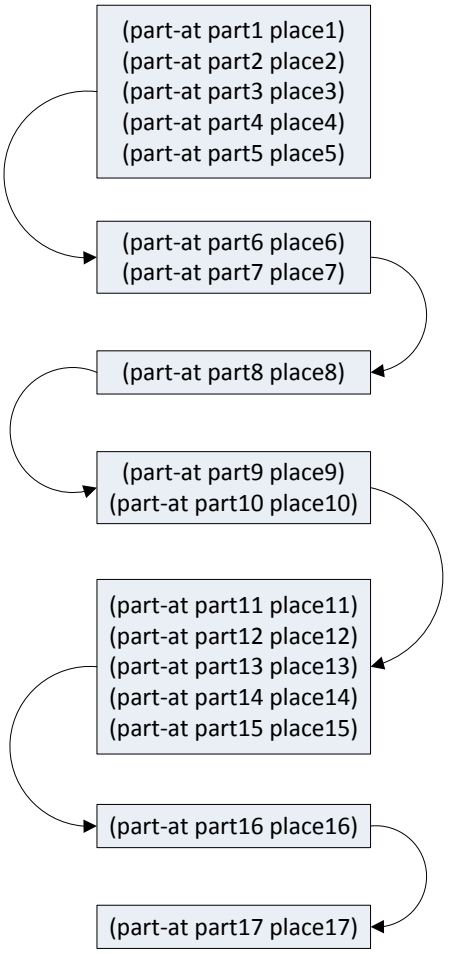
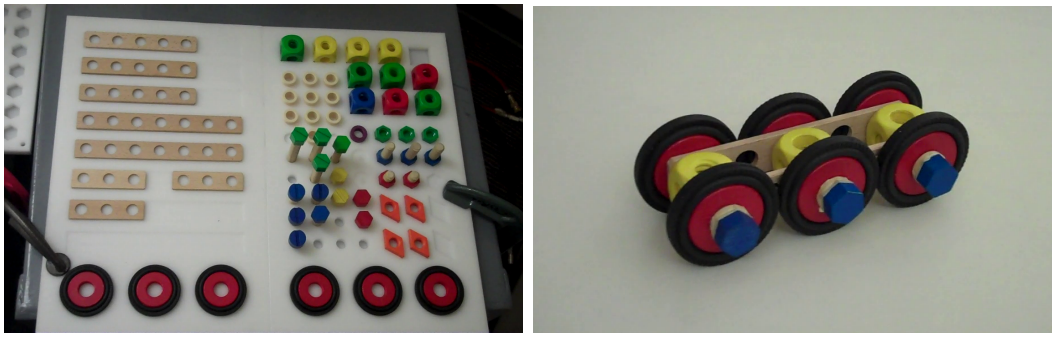



Figure 19: Trajectory constraint diagram for model airplane assembly.

Figure 19 shows the trajectory constraint diagram for the model airplane, or the



(a) Assembly Parts (b) Model Vehicle

Figure 20: Model Vehicle Assembly Task Used to Demonstrate Task Abstraction

sequence requirements that must be met for the airplane to be assembled correctly. Goals within each box can be met in any order (in the case of the top part of the figure, the ordering does not matter for bottom-most parts set into the fixture), but all goals in each box must be met before goals in a subsequent box may be attempted (the fixture parts must be placed before the body and wing parts are laid down.)

#### *4.3.1.2 Vehicle Assembly*

The second task the robot is given to perform is the assembly of a model vehicle, which can be seen in figure 20b. Like the airplane model, the vehicle model is also an appropriate choice for this example because the task required the use of many of the skill primitives specified in the Assembly Taxonomy model. It is a good example to use alongside the model airplane because although they both use similar parts and sequences of assembly actions, the possible plans themselves can take on very different shapes. This model is built using 23 of the wooden construction parts.

Figure 21 shows the trajectory constraints for the vehicle model. These constraints take on a different form for this model compared to the previous model. Where the previous model had very distinct regions where the ordering of the actions did not matter, here there are both loose and strict constraints on ordering. For the top part of the diagram, strict constraints are seen between parts {1,4,7}, {2,5,8} and {3,6,9}. That is, part1 (bolt) must be placed before part4 (corresponding washer),

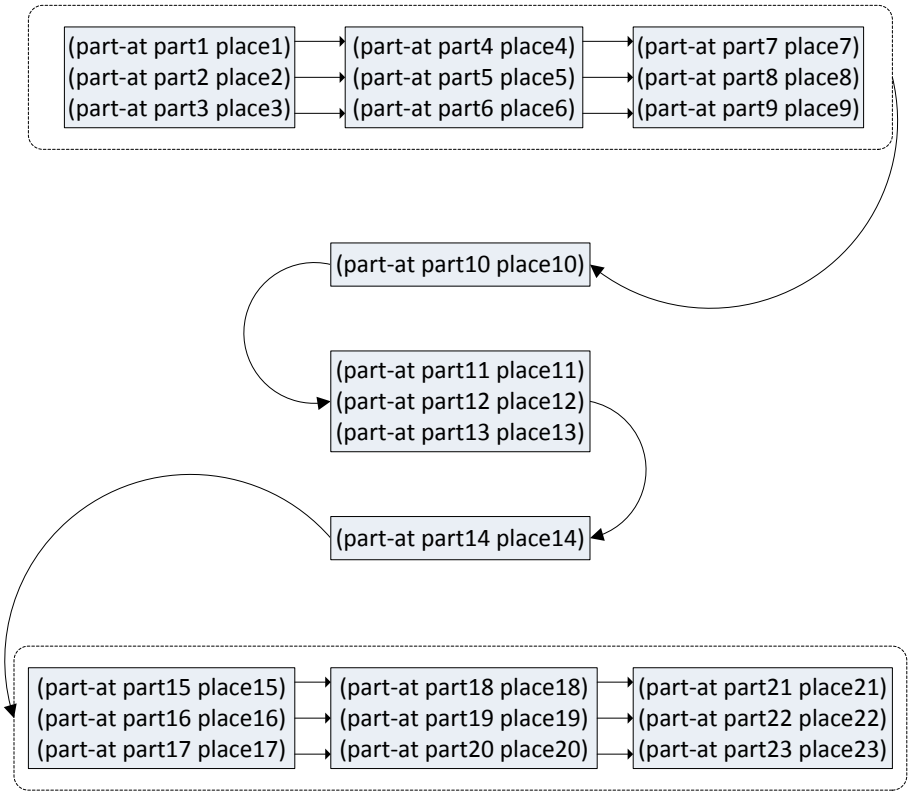



Figure 21: Trajectory constraint diagram for model vehicle assembly.

which must both be placed before part7 (tire), but so long as this ordering is met, it does not matter how that takes place interleaved with the other sub-assemblies (the loose constraint.) The dotted outline indicates that all of these steps, in whatever order they occur, must happen before part10 is placed.

### 4.3.2 Experimental Results

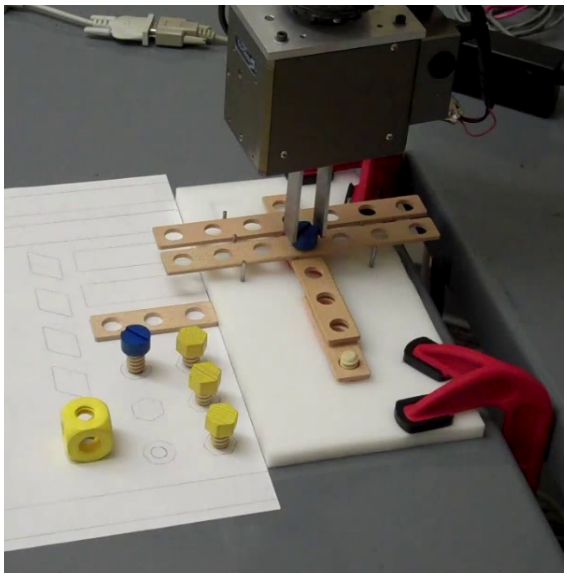


Figure 22: Task abstraction - robot assembly of a model airplane.

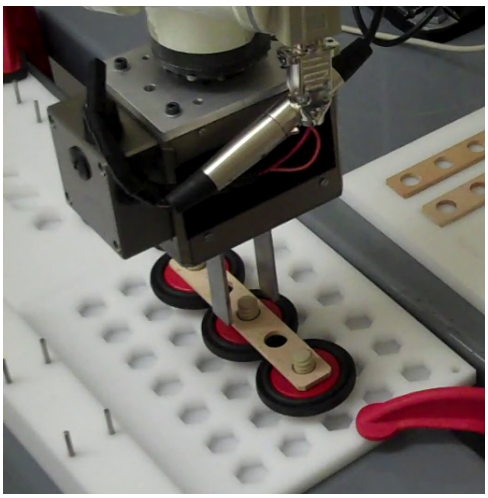
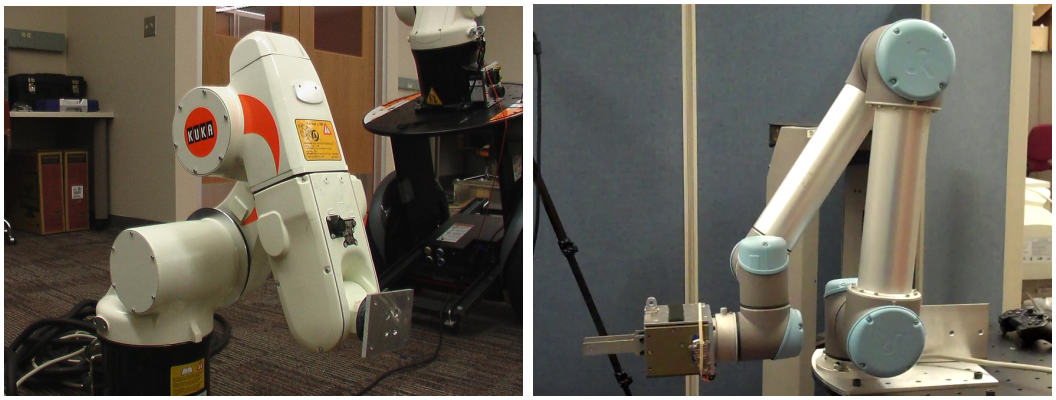


Figure 23: Task abstraction - robot assembly of a model vehicle.

In this experiment the same robot (in this case the KUKA KR5-sixx) was used



(a) Platform 1 - KUKA KR5-sixx (b) Platform 2 - Universal Robots UR5

Figure 24: Platforms Used to Demonstrate Platform Abstraction

to perform two different tasks. The task descriptions used the same abstract skill primitives defined in the task modeling framework. Figures 22 and 23 show the robot successfully assembling each model.

There were a few assumptions that went into the experiment itself. First, we assumed that all desired parts in the part bin are visible and accessible. It was also assumed that perception (object detection and pose estimation) was readily available to the system and accurate. This was simulated using actual poses for the parts for manipulation. Whenever ***Detect*** was used to determine the pose of some desired object for manipulation, instead of receiving the pose estimate by accessing a perception-based pose estimation algorithm, for the sake of the simulation it accessed the ground truth pose determined prior to the experiment.

## *4.4 Platform Abstraction*

### 4.4.1 Experimental Design

The second experiment will demonstrate abstraction across platforms, or that a single task plan, described using skill primitives in the framework, can be used on multiple different platforms to perform the same task.

In this experiment we will use the same robot assembly task on two different robot platforms to demonstrate that the abstractions used in the task modeling framework

are platform agnostic. The task itself is the model airplane assembly task presented in the previous section (see figure 17b). The task plan used can be seen in figure 18.

Figure 24 shows the two robot platforms used to demonstrate platform abstraction. The first platform we are using is a KUKA KR5-sixx (figure 24a). It is a light-weight industrial manipulator, designed and targeted for light manufacturing tasks. The KR5-sixx has six degrees of freedom, a payload 5kg, and a maximum reach of almost 1 meter.

The second robot used in this experiment is a Universal Robots UR5 platform (figure 24b). This robot similarly has a payload of 5kg, a reach of almost 1 meter, and six degrees of freedom, though the configuration of the robot itself is different than that of the KR5-sixx.

For both robots, manipulation is done using a Schunk parallel-plate gripper.

### 4.4.2 Experimental Results

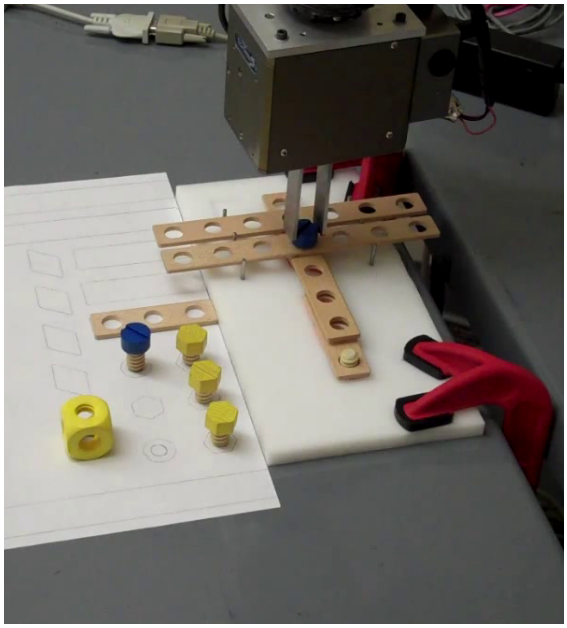


Figure 25: Platform 1 (KR5) performing the assembly task.

Figures 25 and 26 show both robots successfully assembling the model airplane. As described in the previous section, each robot was running the identical task description

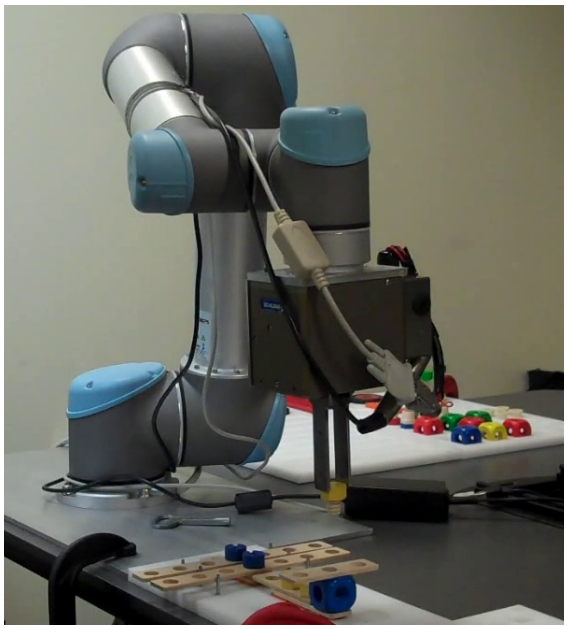


Figure 26: Platform 2 (UR5) performing the assembly task.

plan (shown in the sequence diagram in figure 18).

It should be mentioned that for successful platform abstraction, the assumption is made that for each platform that will be used, there exists a complete set of skill primitives implemented, as described in the taxonomy. While this assumption does require some implementation work prior to execution, this work is proportional to the number of skill primitives in the taxonomy, and need only happen one time for any given platform model. Once the skill primitives exist for the platform, they will simply be reused for each task execution.

## *4.5 Discussion*

In this chapter we have described how we can apply the framework detailed in chapter 3 directly to robot applications, in this case in the robot manufacturing assembly domain. This type of representation presents a number of advantages. Representing robot capabilities at this level of abstraction allows the representation to be independent of both objective and implementation. For example, the ***Transport*** skill simply allows the task model to specify that a movement action must be performed, without

worrying about the details of which algorithm must be used to perform that action, or what the overall objective of the system is.

Using this generalized representation makes it possible to apply these skill primitives across a broad range of problems, from tasks similar or very different from those described in the examples, and makes it possible to describe plans and task objectives in a way that is platform agnostic. Using the representation presented in the framework keeps the abstraction level high enough to generalize across many different problem descriptions as well as across different platforms, as the skill primitives exist above the hardware implementation level. This also frees the end user from having to learn a lower-level description of individual robot capabilities and parameter specifications.

In the previous sections we showed that each of these experiments was successful in that we were able to demonstrate both task abstraction and platform abstraction. The true results of the experiments are that they show that we are able to use this high level abstraction approach to run a multitude of different tasks using these generalized skill primitives, and a single task execution plan on a number of different platforms. This means that given any simple task description, we can execute it on any platform for which the taxonomy-modeled skill primitives have been encoded (as mentioned previously, this encoding need only happen once for any given platform model.) The task description itself could be generated by hand, or automatically through various methods such as planning (which we will go into greater detail about in the next chapter.)

### 4.5.1 Skill Library

Sometimes a robot will come across a collection of capabilities that are often performed together repeatedly, such as those required to pickup a part from the parts bin. In these cases, a high-level capability can be defined that is composed of these

lower-level capabilities, and whose parameters satisfy the parameters of each component capability. An example of this concept is presented here.

**SCREW-INSERT (X,T)**

1: *DETECT*
2: *TRANSPORT(X)*
3: *ALIGN*
4: *INSERT*
5: *SCREW MOTION(T)*
6: *UNGRASP*
7: *RETRACT*

For the operation of taking an object and screwing it in at the appropriate location, the ***Screw-Insert*** composition is defined. In it the necessary capabilities are present, including moving the part to pose $X$ (the insertion point) , and screwing the part in with parameters $T$, which is the torque threshold used to determine when the action is complete. Similarly, the ***Pick*** and ***Place*** compositions can also be defined.

This can be applied to the examples previously presented. For example, assembly of the vehicle model would proceed in much the same way as before except in this case the assembly description is simplified with the use of the defined compositions. The robot ***Picks Up*** the first part, and then ***Places*** it in the fixture. This continues until the robot needs to screw on the block for the wheel assembly. Here the robot ***Picks Up*** the block, and then ***Screws*** it in the correct location on the wheel assembly. When the blocks are finished, the robot reverts back to ***Picking Up*** parts and ***Placing*** them on the fixture, until the last bolts need to be screwed on, when the robot finishes the wheel assembly (and the model) by ***Screwing*** on the bolts.

Each of these newly formed high-level skills can be pulled into our pool of available robot capabilities. With these we can create a Skill Library by combining all of our already existing low-level skill primitives with these high-level compositions. This growing Skill Library forms a much more rich and flexible method for describing robot capabilities, as well as task models and action plans.

## *4.6 Conclusion*

In this chapter we have presented a number of ways the proposed high level task modeling framework based on SysML can be used to enable flexible manufacturing by raising the level of robot programming and interaction. We outlined experiments performed to demonstrate how the framework could be used to abstract robot capabilities across different processes, and independently how a single task description could be platform agnostic and run on multiple robots. We discussed the importance of abstraction in simplifying robot programming and interaction and pointed out how the SysML-based framework is well designed for this type of abstraction. The concept of the Skill Library was also presented and discussed. The next chapter will examine various methods used to generate task descriptions, and work done in automated task description generation through formal planning.

# CHAPTER V

# PLANNING ON THE FRAMEWORK

In the previous chapters we have demonstrated how the GTax knowledge transfer framework can be utilized to model robot assembly tasks at a high level, and enable knowledge transfer in the form of task abstraction and platform abstraction. We have shown how knowledge can be represented in the domain, and how it can be used to build sequence diagrams, or action plans, for how the robot is to achieve the overall objective. These sequence diagrams describe the sequence of skill primitives that the robot needs to execute to successfully achieve the overall task objective. As such, they can grow quite long. For example, the sequence diagram for the model airplane assembly task has almost two hundred and fifty individual skill primitives to execute the task. Creating this sequence diagram by hand can be time consuming and tedious. While using high-level skills such as ***Pick*** and ***Place*** can reduce the size of the sequence diagram (while also reducing time and increasing readability), as assembly tasks scale up beyond putting together simple models, the sequence diagrams can become large and quite complex.

Research has been undertaken to investigate how a planning domain representation in concert with the task modeling framework presented in the previous chapters can be used to automatically generate these execution sequence diagrams, thereby reducing the work involved in plan creation for complex manufacturing tasks [29]. This provides the opportunity to relieve some burden from the robot user, and to improve the reliability of those plans. Given the scope of the overall research, this work focuses on basic functionality of robot manipulators in a manufacturing setting. The investigation shows that recent versions of the so-called Planning Domain Definition

Language (PDDL) [43], and software planners conforming to them, are suitable for representing the relevant problems and searching for solutions in an automated way.

This work defines the BasicAssembly PDDL planning domain that captures the basic capabilities of manufacturing robots. In particular, the domain represents the skill primitives that are abstracted in the capabilities SysML taxonomy described in the previous chapters. These actions are represented in terms of their preconditions and effects. For example, the following statement is an action schema of the BasicAssembly domain that specifies the action of a robotic arm `?a` grasping a part `?p` from station `?s`.

```
(:action grasp
  :parameters (?a - arm ?p - part ?s - station)
  :precondition (and
    (arm-at ?a ?s)
    (part-at ?p ?s)
    (arm-active ?a grip)
    (arm-holding ?a no-part)
    (pose-detected ?a ?p ?s))
  :effect (and
    (arm-holding ?a ?p)
    (not (arm-holding ?a no-part))
    (not (part-at ?p ?s))
    (not (pose-detected ?a ?p ?s)))
)
```

This planning domain has been successfully applied to several variations on an example of a multi-robot system in an automobile manufacturing environment. It is expected that this work will contribute to the manufacturing robotics domain by showing how this task modeling framework can be used in conjunction with formal planning methods to improve the automation process and reduce dependence on user knowledge and effort. To the AI planning community, it is expected that this work

will contribute to the body of knowledge by demonstrating relatively new planning methods, taking into account trajectory constraints, successfully applied to specific manufacturing problems.

In this chapter we present the idea that by using AI planning in concert with formal task modeling, the overhead associated with plan creation for complex tasks can be reduced. The proposed approach uses our SysML taxonomy to model the system capabilities and the process specification, and the PDDL planning language to determine acceptable objective solutions. This idea is applied to the manufacturing domain, and examples are shown modeling a multi-robot system in an automobile manufacturing environment. A discussion is given regarding the merits of the demonstrated approach.

## *5.1 Introduction*

Robotics in manufacturing is seeing a second renaissance of sorts in today's world. This stems in a large part from recent advances in perception and manipulation platforms, and also a desire to free robots from their safety engineered cages to allow humans to work along side them to solve harder problems. Examples include efforts from Rethink Robotics (the Baxter platform) and ABB (the Frida platform), who are designing robots with flexible manipulation and advanced perception, that are specifically designed to work in collaboration with humans in human environments safely. With this renewed desire to automate tasks in the manufacturing domain, a problem that has persisted presents itself again: namely, the best way to model tasks, and to what extent can we automate the modeling of tasks and execution plans in these settings.

This problem applies to both these new areas where robots and humans are collaborating to solve tasks, but also to other areas of robotics and automation in manufacturing, such as large scale airplane wing assembly. One reason it has persisted for

so long is simply that programming robots is hard. It requires deep technical knowledge, it is time consuming, and programming robots is done by specification with respect to the system requirements. For decades researchers in both industry and academia have looked for ways to reduce the overhead that this kind of automation entails. Methods such as programming by demonstration have attempted to reduce this load by teaching a robot tasks by demonstrating how those tasks are performed. Programming in this way requires less technical knowledge, is more intuitive, and would thus cut the necessary time required to program the robot.

Unfortunately, at least in manufacturing, these methods have not yet proven to be robust enough to generalize across a wide enough area such that they can be useful, and we are still left searching for a good way to automate task modeling and robot execution, such that these do not require users with specific technical skills and deep domain knowledge.

To be able to feasibly solve this problem, we first need a method to model the tasks necessary to enable the robot to achieve its programming objective. We begin here by specifying a taxonomy that defines skill primitives, or robot-specific skills that a robot can perform such as pick-up or detect, and constraints related to those skill primitives. Having this taxonomy allows us to specify how task components can be used to build higher-level tasks.

We proceed further to also show how artificial intelligence planning methods can be applied to the manufacturing assembly domain using this task modeling framework. We expect this work will contribute to the manufacturing robotics domain by showing how such a framework can be used in conjunction with formal planning methods to improve the automation process and reduce dependence on user effort. To the planning community, we expect this work to contribute to the body of knowledge by demonstrating relatively new planning methods, taking into account trajectory constraints, successfully applied to specific manufacturing problems.

This chapter will proceed as follows. An overview of related work in robotics and manufacturing will be presented in section 5.2. Following this, an introduction to our task modeling method, classical planning, and the BasicAssembly planning domain used in this work will be given in section 5.3. Sections 5.4 and 5.5 will present different examples of planning with manufacturing problems, along with a discussion of different design choices. We will conclude in section 7 by mentioning future directions for this work.

## *5.2 Related Work*

Given that programming robots is a difficult task requiring considerable technical knowledge, a great deal of prior work has focused on addressing how to make robot programming more intuitive and more accessible to operators with less technical experience. Much of this work has focused on finding efficient and effective methods for representing system knowledge to accomplish this task.

One philosophy has worked on encoding task knowledge as a function of motion. Examples of this type of representation include dynamical systems [30] and object-action complexes [37]. Another philosophy has espoused the view that one can effectively represent important system knowledge symbolically, such as topological task graphs [1]. This symbolic approach assumes that the system has more inherent knowledge (it knows how the relationships between the symbols and physical instantiations behave), while it allows for the modeling of more high-level concepts than motion-based representations. Recent work by Kaelbling and Lozano-Pérez [32] attempts to combine symbolic task planning and geometric motion planning into a single approach.

More similar to our approach, other work has used different knowledge representations to simplify the robot programming problem. The work of Lyons et al. [42] defined a model for robot computation using port automata. Kosecka et al. [35] used

a discrete event systems framework to model tasks and behaviors for robotics. Recent work includes that of Dantam, which takes a grammar-based approach to represent sensorimotor information [12].

There is also work that has been done in representing manufacturing and assembly objectives, such as the application of Petri Nets [55]. Another approach is the work of de Mello and Sanderson [27], which uses AND/OR graphs to enumerate all possible paths through the assembly process to get to the overall objective (e.g. an assembled product.) The paper then proposes to use a graph search algorithm to find an appropriate path through the graph based on specific problem specifications. However, this method does not allow for various goal specifications. Symbolic planning methods such as the one we adopt in this paper are able to provide a broad range goal specification based on sub-goals, trajectory constraints, cost metrics, and soft preferences.

Work by Kress-Gazit, et al., [36], [18] uses linear temporal logic to model task specifications to produce correct robot controllers for different tasks. While this planning representation is able to take advantage of the larger range of goal specifications as opposed to the previous example, it is far more complex than the planning representation used in our work, and perhaps is less likely to have a big impact to the manufacturing domain at this stage.

Similar to our goal, Balakirsky et al. [2] used the OWL ontology to provide a method for structuring knowledge in such a way as to be reusable for different problems, applied to kitting applications. Our work likewise proposes a multi-layered representation, but at a different level of abstraction. Our work also differs fundamentally in that our representation is proposed to improve not only modularity in knowledge, but also usability and intuitiveness for users.

## 5.3 System

### 5.3.1 Models, processes, and sequences for manufacturing

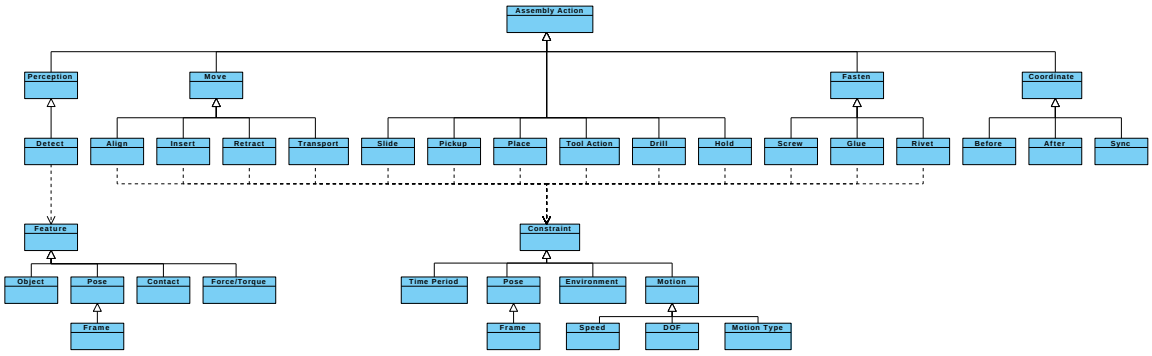



Figure 27: A model instance for assembly tasks.

One goal of this work is to structure knowledge in such a way that lends itself to modularity, such that it can be used in different problems and contexts. To this end, we define the *model space* as the collection of all possible *capabilities* that a robot or robotic system may employ to accomplish some objective in the real world. An instance of model space would model the capabilities required to accomplish tasks in a specific application domain, such as manufacturing, laboratory automation, etc. The model space would thus span all capabilities needed in all robot application domains.

In this work, a *model instance* takes the form of a *SysML taxonomy* [46] that specifies each of these capabilities. SysML (or Systems Model Language) is a general modeling language for systems and systems engineering, and is defined as an extension of the popular UML modeling language. Benefits of using SysML include an expressive modeling language and tools for code generation, verification, and validation of system models.

For example, Figure 27 shows a model space instance for the manufacturing assembly domain [28]. The simplest building blocks in the taxonomy are the *skill primitives*, which are atomic actions that a robot is able to perform. The skill primitives are complemented by certain parameters or constraints on the action to be performed. For

example, actions such as ***Align*** and ***Transport*** will often be done with respect to some goal or destination pose.

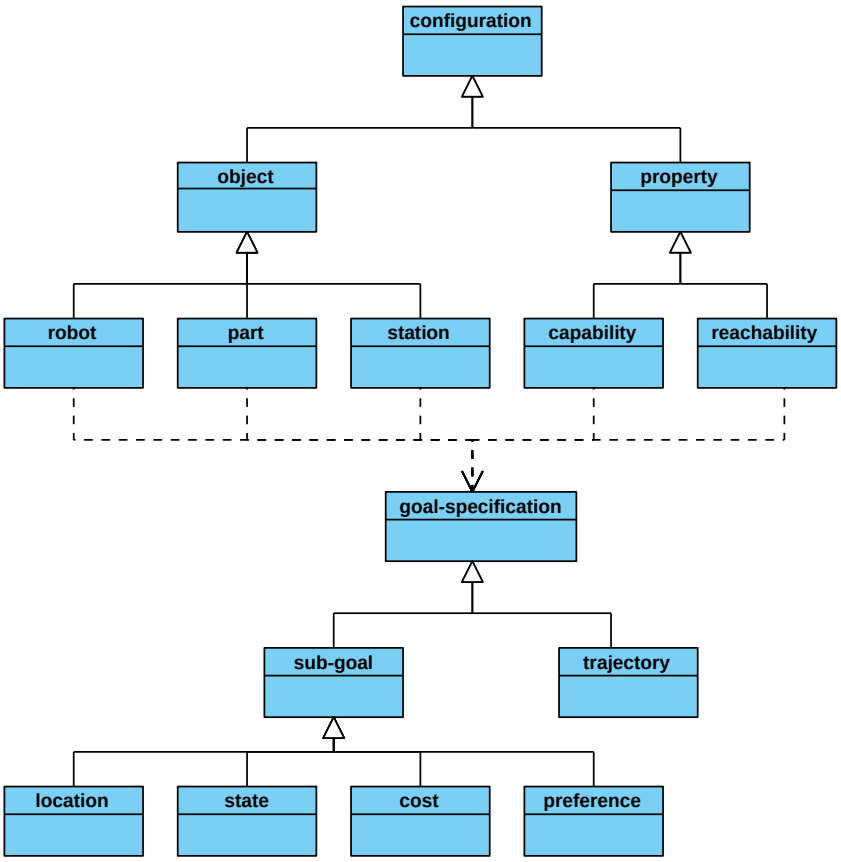



Figure 28: A process instance for an assembly task.

Based on the capabilities specified in the model space, we further define the *process space* as the collection of all system configurations and goal specifications that specify an environmental robot setup and objective. As with model space, a *process instance* takes the form of a SysML taxonomy, and it specifies the robot configurations and goals for a desired application domain. Figure 28 shows an instance for the manufacturing assembly domain.

Using these two instances, we can fully specify a *system* and *objective*. At this point, various methods can be used to find a solution or plan to solve this problem. One method of realizing this plan is the direct method, or having a human directly encode the plan into SysML, which would then specify how the available robots

would execute the actions to solve the problem. In this case the plan takes the form of a SysML *sequence instance*, which uses the hardware dependent implementation of skills to instruct the robot how to go about solving the problem, as for instance in Figure 29.

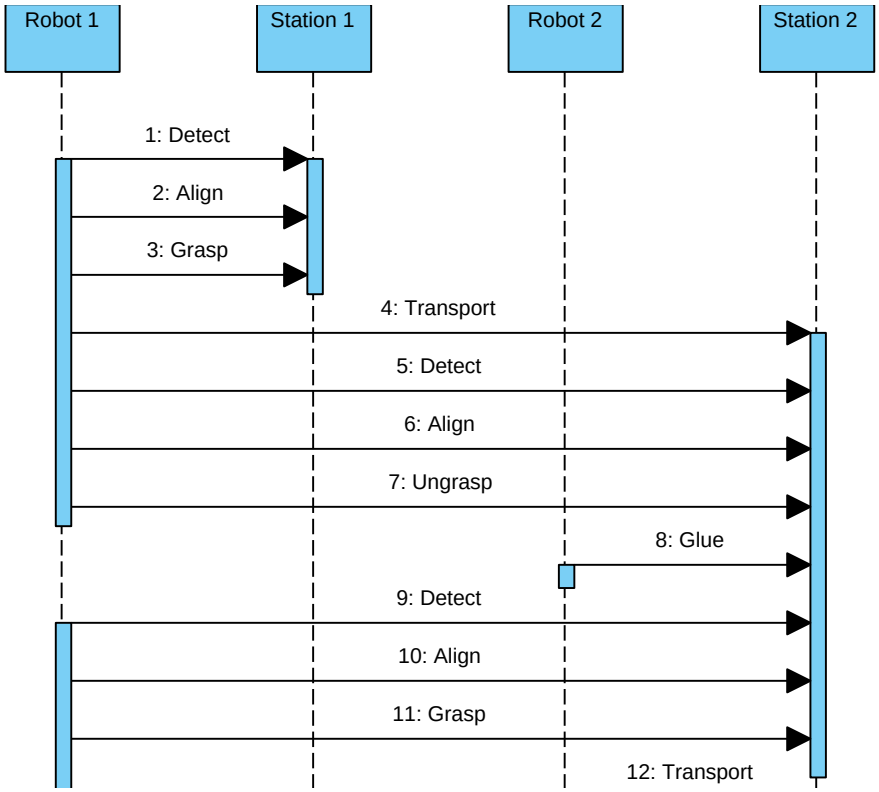



Figure 29: A sequence instance for an assembly task.

The sequence instance is intended to show the organization of an assembly objective into a *sequence of actions* that are to be performed, utilizing the skills described in the model taxonomy as discrete steps in the sequence. Each message from the robot to either the fixture or part bin represent an instantiation of a skill primitive (e.g., message 1 is a ***Detect***).

This manual encoding method can be effective, but has several drawbacks. As mentioned in the introduction, programming robots is hard as it requires deep technical knowledge and is time consuming. Apart from the overhead of specifying an

initial procedure, one obvious difficulty is that any time some change in the configuration (or the target outcome) is required, additional resources are needed in order to specify an updated procedure.

Moreover, as this approach typically does not rely on a formal specification of the preconditions and effects of actions, no automated way can be used to check for errors and inconsistencies in the plan. This makes it possible that a user may even design a plan that could damage a workpiece, or put a robot into a dangerous configuration. Similarly, there is no easy way to guarantee success of the procedure in terms of meeting all of the objective goals or identify any guarantees on the optimality of the procedure. All of these issues are typically resolved by trial and error or using domain-specific tools for simulating and analyzing execution.

This is where artificial intelligence planning methods can offer significant benefits. By integrating planning into the workflow, the *planner* can take a considerable burden off of the user by addressing the previous issues. In this way parametrized solutions can be generated by the system, reducing the overhead required for manually specifying complex manufacturing tasks. Alternatively, the planner can aid the user by presenting potential plans that the user can then further tailor to meet changing needs in dynamic scenarios. In this work we aim for such functionality and adopt the so-called *Planning Domain Definition Language (PDDL)* [43].

The idea is that based on a *model instance* that corresponds to a particular manufacturing domain, we can specify a *PDDL planning domain* that formalizes the preconditions and effects of the available actions in the domain. Then, from a *process instance* that corresponds to a particular manufacturing problem in this domain, we can specify a *PDDL planning problem* that formalizes the configuration of the available robots and resources as well as the intended goal we want to achieve. Then with these two specifications at hand we can employ *PDDL planners* to search and find a solution, that is, in effect an appropriate *sequence instance* for this manufacturing

domain/problem combination.

As the academic field of automated planning evolves, this approach to generating sequence instances in an automated way can also enjoy the increasing set of features for the developed PDDL planners. In particular, apart from generating a sequence instance that satisfies basic requirements, state-of-the-art planners can take into account action cost, duration, parallelism, and are able to search for optimized solutions that maximize or minimize given metrics and preferences.

Next, we introduce the necessary details of STRIPS planning and PDDL, and proceed to specify a PDDL planning domain for the assembly manufacturing domain.

### 5.3.2 STRIPS planning and the language of PDDL

In the area of classical planning one is faced with the following task: given i) a complete specification of the initial state of the world, i.e., an application domain, ii) a set of action schemas that describe how the world may change, and iii) a goal condition, one has to find a sequence of actions such that when applied one after the other in the initial state, they transform the state into one that satisfies the goal condition. In this paper we focus on the STRIPS formalism [17] with some extensions.

In STRIPS, the representation of the initial state, the action schemas, and the goal condition is based on literals from predicate logic. For example, $\textit{arm-at}(\textit{arm}_3, \textit{station}_7)$ is a positive literal that may be used to represent that the robotic arm "3" is located at the working station "7". Similarly, the negative literal $\neg\textit{arm-holding}(\textit{arm}_3, \textit{part}_{12})$ may be used to say that the same arm is not holding a particular part.

The *initial state* is specified as a set of positive ground literals. This provides a complete specification of the state based on a closed-world assumption, that is, for all ground literals not included in the set, the negative version of the literal is assumed to hold.

The *action schemas* specify the available actions as well as their preconditions

and effects using sets of ground literals. In the case of preconditions a set of positive literals specify what needs to be present in the state representation in order for the action to be executable, and for the effects of an action, a set of positive and negative literals specify how the state should be transformed after action execution: all the positive literals in the set of effects are added in the set describing the state, and all negative literals are removed.

A *goal condition* is also a set of positive ground literals. The intuition is that the goal is satisfied if all the literals listed in the goal condition are included in the set that describes the state. A solution then to a planning problem is a sequence of actions such that if they are executed starting from the initial state, checking for corresponding preconditions, and applying the effects of each action one after the other, they lead to a state satisfying the goal condition.

In this paper we focus on a specific syntax for describing STRIPS planning tasks following the Planning Domain Definition Language (PDDL) [43]. PDDL is a standard language for specifying planning tasks that is widely used in the academic planning community. In PDDL, the specification of the predicates and the action schemas is separated from the specification of the initial state and the goal condition. The first part is typically referred to as the *planning domain* and the second part as the *planning problem*. This allows us to define a number of planning problems for the same planning domain. In particular, for a manufacturing application domain a planning domain can specify the functionalities of robots that may be available at different times. Using this planning domain then, planning problems can represent different configurations for the available robots and a goal condition for the corresponding manufacturing task.

The syntax of PDDL follows the logical representation of literals but a prefix notation is used. In this way, the positive literal *arm-holding*$(a, p)$ may be represented as `(arm-holding ?a ?p)`, and the negative version of the literal may be represented

as `(not(arm-holding ?a ?p))`. Note that variables are denoted using a preceding question mark. The special predicates `(:predicates ...)` and `(:action ...)` are used for the specification of the planning domain, with the intuitive meaning. Similarly, the special predicates `(:objects ...)`, `(:init ...)`, and `(:goal ...)` are used to specify a planning problem. This notation will become more clear in the next section where we introduce the BasicAssembly planning domain.

Finally, we will appeal to some more advanced features of PDDL that go beyond STRIPS. In particular we will adopt *types* for objects following functionality ADL [48]. For example `?a - arm` will be used to denote that variable `?a` can only be replaced by an object of type `arm`. Moreover, we will appeal to PDDL3 [23], one of the latest specifications of the PDDL language, in order to specify simple *trajectory constraints* for the desired goal condition. These constraints will be used to specify the intended order in which the sub-goals of the manufacturing task need to be executed, essentially specifying the steps of a process to be planned.

### 5.3.3 The BasicAssembly planning domain

In this section we present the BasicAssembly PDDL planning domain that captures some of the basic capabilities of manufacturing robots. In particular, the domain represents the high-level actions that are abstracted in the Capabilities SysML taxonomy described in the previous section. These actions are represented in terms of their preconditions and effects. The intention is that in this way a system can plan on the level of these capabilities of robots, leaving lower-level activities such as motion planning to be handled at a lower level at run-time.

A manufacturing site is represented as a set of robotic *arms* that abstract the sensors and actuators of the site, a set of *parts* that abstract the pieces to be handled in order to achieve the relevant manufacturing task, and a set of *stations*, each of which may be occupied by one or more robotic arms and may be used to place or

hold a number of parts. Except for the very basic actions of moving and grasping that are modeled with a separate action, a set of *tools* represents the available operations that can be performed on the parts. In PDDL terms, arms, parts, stations, and tools are all objects of the PDDL problem. Following the typed-approach though, there are four basic types in order to distinguish between them, namely types `arm`, `part`, `station`, and `tool`.

We assume that the low-level details required for an arm interacting with parts and stations is abstracted by two high-level "detect-pose" actions, which can be handled appropriately by each robotic arm at execution time. In the high-level representation of BasicAssembly, in order for a robotic arm to pick up a part `?p` located at station `?s`, it should first move to station `?s`, perform a detect-pose action using available sensors, and then grasp the part using an attached gripper (provided the arm is equipped with one).

The BasicAssembly PDDL domain uses a number of predicates and action schemas to represent some basic functionalities in manufacturing domains, as explained next.

### The predicates of the domain

The predicates of the BasicAssembly planning domain essentially represent properties of the state of the manufacturing site at any time. In particular, the following predicates represent information regarding the available robotic arms.

- `(arm-canreach ?a - arm ?s - station)`
- `(arm-at ?a - arm ?s - station)`
- `(arm-capabilities ?a - arm ?t - tool)`
- `(arm-active ?a - arm ?t - tool)`
- `(arm-holding ?a - arm ?p - part)`

The following predicates represent properties of the parts. Note that some literals are used as book-keeping information for modeling the robotic arm actions and their effects.

- `(part-at ?p - part ?s - station)`
- `(part-state ?p - part ?t - tool)`
- `(pose-detected ?a - arm ?p - part ?s - station)`

More details for these predicates and their use can be found in the complete PDDL specification of the BasicAssembly domain in the appendix.

The intuition is that a set of positive ground literals of this kind can be used to represent the initial state of the manufacturing site and the goal condition for planning purposes. For example, the positive ground literals `(arm-canreach arm1 st1)` and `(arm-canreach arm1 st2)` can be used to model that `arm1` can reach both stations `st1` and `st2` (and in fact according to the closed world assumption *only those stations*). Similarly, `(arm-capabilities arm1 grip)` may be used to model that `arm1` has a gripper.

We now proceed to present the action schemas that characterize the available actions for the robotic arms.

### The actions of the domain

Action schemas in PDDL are defined in terms of their preconditions and effects using the predicates of the domain. For example, the following statement is an action schema of the BasicAssembly domain that specifies the action of a robotic arm `?a` grasping a part `?p` from station `?s`.

```
(:action grasp
  :parameters (?a - arm ?p - part ?s - station)
```

```
  :precondition (and
    (arm-at ?a ?s)
    (part-at ?p ?s)
    (arm-active ?a grip)
    (arm-holding ?a no-part)
    (pose-detected ?a ?p ?s))
  :effect (and
    (arm-holding ?a ?p)
    (not (arm-holding ?a no-part))
    (not (part-at ?p ?s))
    (not (pose-detected ?a ?p ?s)))
)
```

The intuition is that a ground instance of this action schema, e.g., `(grasp arm1 p station2)`, can be performed when the arm is located at the same station as the part, the grip tool is activated, the arm is not holding another part, and the required information about pose detection is already present. These requirements can be met by the previous execution of other appropriate actions, in particular one that moves the arm to the right station, one that activates the gripper, and one that provides information about the low-level pose to be realized. These actions are also part of the BasicAssembly domain and will be described next.

The result of performing the action is that the relevant part is no longer located at the station, therefore the corresponding literal is removed from the description of the state, while a positive literal is added to represent that the arm is now holding the part, among a couple of other details.

Note that in order to allow maximum compatibility with available planners we use only positive literals in the preconditions of actions, and as a result we model the information that a robotic arm `?a` is not holding any part with the literal `(arm-holding ?a no-part)`. This could be done also with another predicate of the form `(arm-gripperfree ?a)` but we chose to use the same predicate `arm-holding`

for uniformity. Note also that we abstract the pose detection and alignment needed, using an appropriate high-level condition concerning the arm, part, and station in question, that is, `(pose-detected ?a ?p ?s)` that can be achieved by an appropriate detect action as explained next.

We now give a list of the action schemas of the BasicAssembly domain which have the intuitive meaning. The details for each action schema (parameters, preconditions, effects) can be found in the complete PDDL specification of the BasicAssembly domain in the appendix.

- `activate(?a -arm ?old - tool ?new - tool)`
- `grasp(?a - arm ?p - part ?s - station)`
- `ungrasp(?a - arm ?p - part ?s - station)`
- `move(?a - arm ?from - station ?to - station)`
- `carry(?a - arm ?p - part ?from - station ?to - station)`
- `employ(?a - arm ?t - tool ?p - part ?s - station)`
- `detect-pose-part(?a - arm ?p - part ?s - station)`
- `detect-pose-station(?a - arm ?p - part ?s - station)`

Note that in order to allow maximum compatibility with available planners we also avoided using conditional effects, and as a result chose to model the movement of an arm by two different actions, one when the gripper is empty (`move`) and one when the arm holds some part (`carry`).

## *5.4 A concrete example using BasicAssembly*

In this section we will present an application example that demonstrates the use of the BasicAssembly planning domain.

### 5.4.1 Car-door manufacturing scenario

This task concerns the preparation of a car-door part for assembly into a car, taken directly from the factory floor. We will present two variations on the common scenario according to which the part must be glued and then welded before it is added to the assembly line.

### 5.4.2 Process instance P1

The first configuration of the scenario is shown in Figure 30. There are four stations: the part bin (the originating location for the parts to be worked on), work station 1 (the gluing station), work station 2 (the welding station) and the assembly line. There are three robots, one which is able to reach all stations, and two specialized robots for each work station. This problem is formalized in PDDL as follows:

```
(define (problem p1) (:domain assembly)
  (:objects
    arm1 arm2 arm3 - arm
    car-door no-part - part
    part-bin station1 station2 assembly-line - station
    grip detect weld glue no-tool - tool )

  (:init
    (arm-canreach arm1 part-bin)
    (arm-canreach arm1 station1)
    (arm-canreach arm1 station2)
    (arm-canreach arm1 assembly-line)
    (arm-canreach arm2 station1)
    (arm-canreach arm3 station2)
    (arm-at arm1 part-bin)
    (arm-at arm2 station1)
    (arm-at arm3 station2)
    (arm-capabilities arm1 grip)
    (arm-capabilities arm1 detect)
    (arm-capabilities arm2 glue)
    (arm-capabilities arm2 detect)
    (arm-capabilities arm3 weld)
    (arm-capabilities arm3 detect)
    (arm-active arm1 no-tool)
    (arm-active arm2 no-tool)
    (arm-active arm3 no-tool)
    (arm-holding arm1 no-part)
    (arm-holding arm2 no-part)
    (arm-holding arm3 no-part)
    (part-at car-door part-bin))

  (:goal
```

```
    (and (part-state car-door glue)
         (part-state car-door weld)
         (part-at car-door assembly-line)))

  (:constraints
    (and (sometime-before (part-state car-door weld)
         (part-state car-door glue))
              (sometime-before (part-at car-door assembly-line)
         (part-state car-door weld)))))
```

Predicate `:objects` is used to specify all the available objects in the PDDL problem (corresponding to the process instance in question). Note that objects are listed according to their type, e.g., here there objects of type `arm` are specified.

Predicates `:init` and `:goal` specify the initial state and the desired final state, using positive ground literals. The initial state is specified as a list of literals, while the goal condition is formalized as a logical sentence (in this case using logical conjunction). The intuition is that the final state should be such that all three sub-goals hold, i.e., the glue tool has been employed to the car-door, the weld tool also, and the car-door is located at the assembly line.

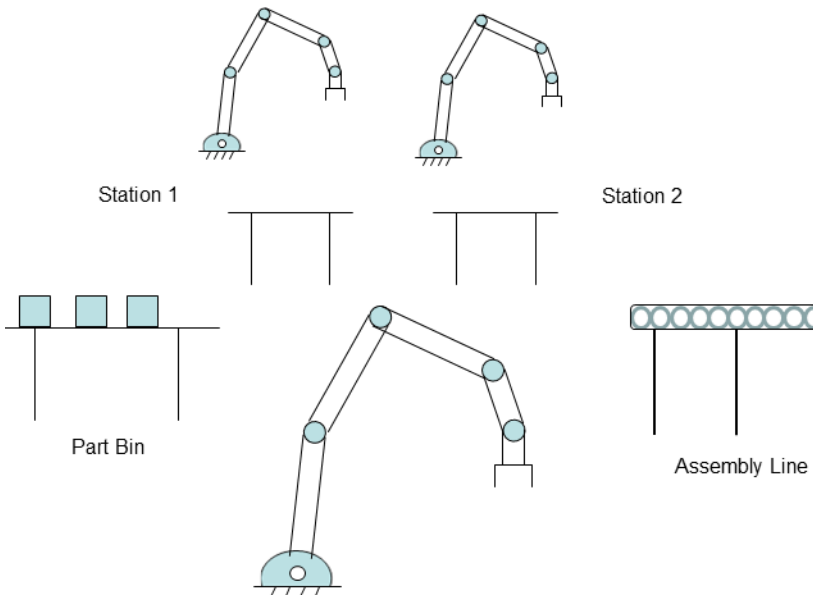



Figure 30: Assembly process instance P1.

Note that in basic STRIPS planning problems there is no information about the

order in which the sub-goals need to be achieved. Extensions, though, have investigated how to fine-tune the resulting plan in terms of soft preferences and hard constraints. In particular in PDDL3, one of the latest versions of PDDL, hard trajectory constraints can be used to specify the order in which sub-goals should be achieved. These are specified using the `:constraints` predicate and the `sometime-before` keyword.

Trajectory constraints are a crucial aspect in the context of manufacturing as essentially we are interested in planning for a particular type of sequence, not just any one that achieves sub-goals in any order. As this feature is a recent addition, currently not many planners fully support it. For our experiments we used the planner OPTIC [3] which was one of the very few able to handle trajectory constraints well and also supports other features we intend to use.

The resulting plan is shown here in time-steps:

```
01: (activate arm1 grip detect)
01: (activate arm2 glue detect)
01: (activate arm3 weld detect)
02: (detect-pose-part arm1 car-door part-bin)
03: (activate arm1 detect grip)
04: (grasp arm1 car-door part-bin)
05: (carry arm1 car-door part-bin station2)
05: (activate arm1 grip detect)
06: (detect-pose-station arm1 car-door station2)
07: (carry arm1 car-door station2 station1)
08: (detect-pose-station arm1 car-door station1)
09: (carry arm1 car-door station1 assembly-line)
10: (detect-pose-station arm1 car-door assembly-line)
11: (activate arm1 detect grip)
11: (carry arm1 car-door assembly-line station1)
12: (ungrasp arm1 car-door station1)
13: (detect-pose-part arm2 car-door station1)
13: (activate arm1 grip detect)
14: (activate arm2 detect glue)
14: (detect-pose-part arm1 car-door station1)
15: (employ arm2 glue car-door station1)
15: (activate arm1 detect grip)
16: (grasp arm1 car-door station1)
17: (carry arm1 car-door station1 station2)
18: (ungrasp arm1 car-door station2)
19: (detect-pose-part arm3 car-door station2)
19: (activate arm1 grip detect)
20: (activate arm3 detect weld)
20: (detect-pose-part arm1 car-door station2)
21: (employ arm3 weld car-door station2)
```

```
21: (activate arm1 detect grip)
22: (grasp arm1 car-door station2)
23: (carry arm1 car-door station2 assembly-line)
24: (ungrasp arm1 car-door assembly-line)
```

Note that the planner accounts for parallelism as some actions of different arms occur at the same time, e.g., the activation of tools for all three arms at time-step 01.

### 5.4.3 Process instance P2

Now we report on a variation that is similar to the previous process instance, except that this time robot `arm1` is not able to reach `station2`. Instead, the specialized robot `arm2` is able to reach both `station1` and `station2`. This implies that moving the car-door part from `station1` to `station2` to `station3` cannot be done by `arm1` alone as in the previous case. Instead, an additional coordination between `arm1` and `arm2` is required. All other considerations are the same as the previous example. The resulting plan follows:

```
01: (activate arm1 grip detect)
01: (activate arm2 grip detect)
01: (activate arm3 weld detect)
02: (detect-pose-part arm1 car-door part-bin)
03: (activate arm1 detect grip)
04: (grasp arm1 car-door part-bin)
05: (carry arm1 car-door part-bin station1)
05: (activate arm1 grip detect)
06: (detect-pose-station arm1 car-door station1)
07: (carry arm1 car-door station1 assembly-line)
08: (detect-pose-station arm1 car-door assembly-line)
09: (activate arm1 detect grip)
09: (carry arm1 car-door assembly-line station1)
10: (ungrasp arm1 car-door station1)
11: (activate arm1 grip detect)
11: (detect-pose-part arm2 car-door station1)
12: (detect-pose-part arm1 car-door station1)
12: (activate arm2 detect glue)
13: (activate arm1 detect grip)
13: (employ arm2 glue car-door station1)
14: (activate arm2 glue grip)
15: (activate arm2 grip detect)
16: (detect-pose-part arm2 car-door station1)
17: (activate arm2 detect grip)
18: (grasp arm2 car-door station1)
19: (activate arm2 grip detect)
20: (detect-pose-station arm2 car-door station1)
21: (carry arm2 car-door station1 station2)
```

```
22: (detect-pose-station arm2 car-door station2)
23: (activate arm2 detect grip)
24: (ungrasp arm2 car-door station2)
25: (detect-pose-part arm3 car-door station2)
25: (activate arm2 grip detect)
26: (activate arm3 detect weld)
26: (detect-pose-part arm2 car-door station2)
27: (employ arm3 weld car-door station2)
27: (activate arm2 detect grip)
28: (grasp arm2 car-door station2)
29: (carry arm2 car-door station2 station1)
30: (ungrasp arm2 car-door station1)
31: (grasp arm1 car-door station1)
32: (carry arm1 car-door station1 assembly-line)
33: (ungrasp arm1 car-door assembly-line)
```

Note that this process instance needs more time-steps than the previous one. This variation demonstrates the planner's ability to deal with configuration and process changes.

## *5.5 Discussion*

In this work we illustrate the utility of using a formal planning domain specification coupled with a task modeling framework. The simple example presented highlights some of the basic aspects of what can be achieved.

As a first step we focused on a high-level representation of the preconditions and effects of actions in a manufacturing assembly domain, that can be used to provide well-structured solutions in an automated way. These solutions still require manual work in a lower level in order to connect the high-level abstract actions to the low-level capabilities of each component. In the general case of manual robot programming, this is difficult and requires resources such as time and technical knowledge. When the scenario is updated or changed, the manual robot programmer must reprogram the entire scenario from the ground up, even for small changes. The intention is that by relying on an appropriate abstraction for robotic actions and realizing the high-level/low-level connection for the available hardware, one can benefit by re-using these models to handle future scenarios, and reduce the resources required to realize

the solutions.

It should be noted that a number of assumptions go into this approach as currently implemented. The lowest level of abstraction relies on hardware-specific implementations such as motion planners and pose estimation algorithms. The current system does not take fault detection or recovery into consideration, and assumes that actions proposed by the system are successfully achieved by the robot. Other assumptions have to do with the environment, or process model. For example, we believe it is not unreasonable to assume that every workpiece in the environment is reachable, as in manufacturing settings the environment is often highly engineered to make the task achievable. Note also that this approach will not significantly speed up some aspects of programming the system, such as specifying complex kinematic motions (the weld trajectory, for example). This is to be expected, however, as this approach has targeted modeling and generation of manufacturing assembly solutions, not methods of parameterization.

One direction for future work is to investigate more refined abstractions that can provide a variable level-of-detail for available hardware platforms used in practice. For example, it may be helpful (or for some platforms necessary) to distinguish as two separate actions the task of detecting an object and detecting a location pose for placing that object. Also, different types of robotic devices may be able to handle an action set that spans more than one level-of-detail. The intuition is that as the action representations become more specific modeling the capabilities of existing platforms, the planner obtains more detailed information and can then provide solutions of better quality that can be incorporated easier in the real manufacturing setting.

A different direction for future work is incorporating more features for refining the desired solution for a sequence instance. PDDL3 already accounts for cost metrics, preferences, as well as continuous time and durative actions. We intend to investigate which combination of features is intuitive and practical and can be effective in

providing greater flexibility by automatically generating solutions as sequences for complex manufacturing problems.

Finally, our experimentation with available PDDL planners showed that even though many advanced features have been investigated leading to robust solutions, there is less focus on trajectory constraints, which is the most crucial one for the type of problems we are interested in the manufacturing setting. We believe that our work may provide useful feedback to the planning community with respect to a possible wide application domain.

## *5.6 Conclusion*

In this chapter we investigate how a planning domain representation in concert with a task modeling framework can be used to reduce the work involved in plan creation for complex manufacturing tasks. This provides the opportunity to relieve some burden from the robot user, and to improve the reliability of those plans. For the scope of this paper we focused on basic functionality of robotic arms in a manufacturing assembly setting. Our investigation shows that recent versions of the so-called Planning Domain Definition Language (PDDL), and software planners conforming to them, are suitable for representing the relevant problems and searching for solutions in an automated way. For future work we intend to explore more complex manufacturing objectives, especially cases that require more varied specification using metrics and preferences.

### 5.6.1 BasicAssembly PDDL domain

This section presents the BasicAssembly PDDL domain in its entirety.

```
(define (domain assembly)
 (:requirements :strips :typing)

 (:types arm station tool part)

 (:constants
  no-tool grip detect - tool
  no-part - part )
```

```
(:predicates
 (arm-canreach ?a - arm ?s - station)
 (arm-at ?a - arm ?s - station)
 (arm-capabilities ?a - arm ?t - tool)
 (arm-active ?a - arm ?t - tool)
 (arm-holding ?a - arm ?p - part)
 (part-at ?p - part ?s - station)
 (part-state ?p - part ?t - tool)
 (pose-detected ?a - arm ?p - part ?s - station) )

(:action activate
 :parameters (?a -arm ?old - tool ?new - tool)
 :precondition (and (arm-capabilities ?a ?new)
                    (arm-active ?a ?old))
 :effect (and (arm-active ?a ?new) (not (arm-active ?a ?old))) )

(:action detect-pose-part
 :parameters (?a - arm ?p - part ?s - station)
 :precondition (and (arm-at ?a ?s) (arm-active ?a detect)
                    (part-at ?p ?s))
 :effect (pose-detected ?a ?p ?s) )

(:action detect-pose-station
 :parameters (?a - arm ?p - part ?s - station)
 :precondition (and (arm-at ?a ?s) (arm-active ?a detect)
                    (arm-holding ?a ?p))
 :effect (pose-detected ?a ?p ?s) )

(:action grasp
 :parameters (?a - arm ?p - part ?s - station)
 :precondition (and (arm-at ?a ?s) (arm-active ?a grip)
       (arm-holding ?a no-part) (part-at ?p ?s)
       (pose-detected ?a ?p ?s))
 :effect (and (arm-holding ?a ?p) (not (arm-holding ?a no-part))
       (not (part-at ?p ?s)) (not (pose-detected ?a ?p ?s))) )

(:action ungrasp
 :parameters (?a - arm ?p - part ?s - station)
 :precondition (and (arm-at ?a ?s) (arm-active ?a grip)
                    (arm-holding ?a ?p) (pose-detected ?a ?p ?s))
 :effect (and (arm-holding ?a no-part) (not (arm-holding ?a ?p))
       (part-at ?p ?s) (not (pose-detected ?a ?p ?s))) )

(:action move
 :parameters (?a - arm ?from - station ?to - station)
 :precondition (and (arm-at ?a ?from) (arm-canreach ?a ?to)
       (arm-holding ?a no-part))
 :effect (and (arm-at ?a ?to) (not (arm-at ?a ?from))) )

(:action carry
 :parameters (?a - arm ?p - part ?from - station ?to - station)
 :precondition (and (arm-at ?a ?from) (arm-canreach ?a ?to)
                    (arm-holding ?a ?p))
 :effect (and (arm-at ?a ?to) (not (arm-at ?a ?from))) )

(:action employ
 :parameters (?a - arm ?t - tool ?p - part ?s - station)
 :precondition (and (arm-at ?a ?s) (arm-active ?a ?t)
               (arm-holding ?a no-part) (part-at ?p ?s)
```

```
                (pose-detected ?a ?p ?s))
  :effect (and (part-state ?p ?t) (not (pose-detected ?a ?p ?s))))
)
```

# CHAPTER VI

# INTEGRATION & ANALYSIS

In the previous chapters in this dissertation we have presented an approach to knowledge transfer using SysML as a language for modeling robot capabilities in specific application domains. We have also showed how this could be applied to various problems in skill abstraction and high-level robot interaction. We will now discuss how these aspects can be integrated into a single approach.

## *6.1 Framework Integration*

Given the material presented in the previous chapters regarding knowledge transfer, one question that should be addressed is how can we pull all of these different aspects together into a single approach that can address the problems discussed in the introduction. Namely, how can this knowledge transfer framework allow us to simplify adjustment to known manufacturing process, including changes to the process itself as well as the hardware performing the task. Figure 31 shows how this integrated approach would look.

First the application domain would need to be modeled using the GTax SysML framework. Given that different tasks exist in the domain, it should be possible to specify these tasks using the model. The next step is using these tools to perform this specification. Here we will use the PDDL planning presented in the previous chapter to automatically generate an action plan for achieving the desired goal task. The domain model would need to be translated into a PDDL domain description file, while the task itself would be described using a PDDL problem file. Once this is completed, a PDDL planner could be used to generate the action plan for the task. This plan is then translated into a SysML sequence diagram.

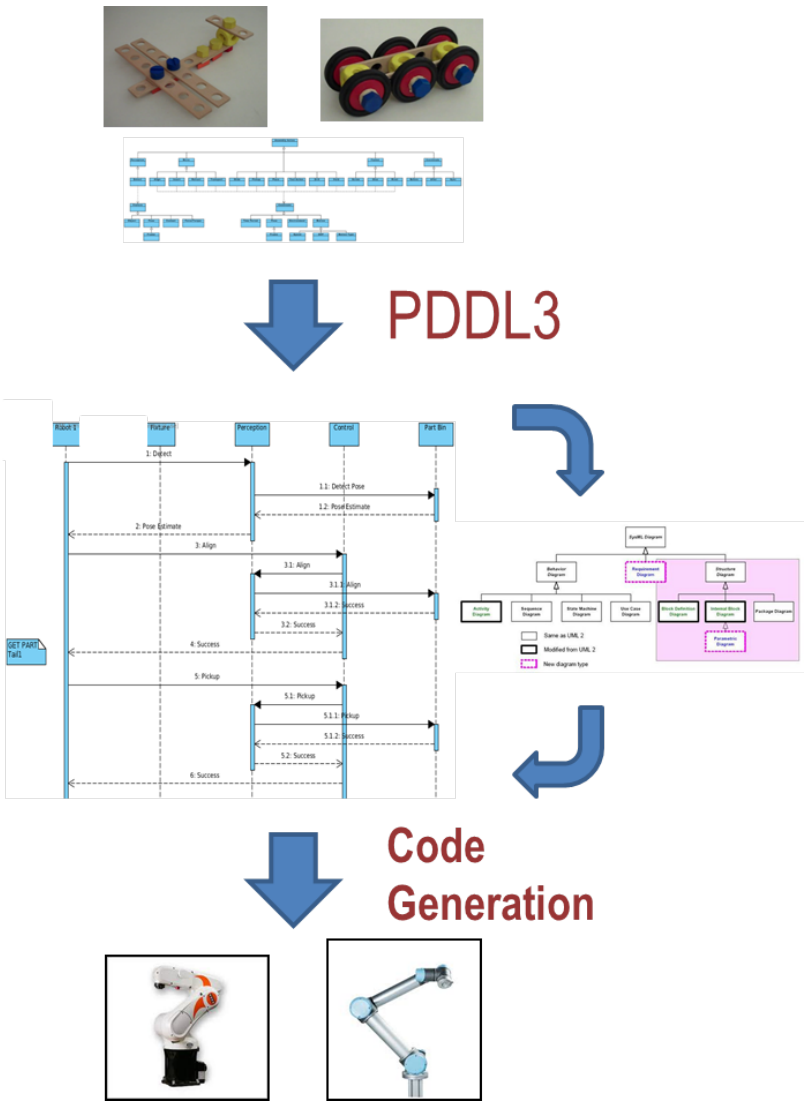



Figure 31: GTax Framework Integration.

Once the sequence diagram for the objective task as been generated, we are then able to use the tools available to SysML, such as verification and validation. Once the model has been verified to meet all of the specified requirements, automatic code generation can then be used to generate code that can be compiled and executed.

From here we can see how platform abstraction is enabled using this approach. When the sequence diagram code is run, it makes calls to generalized skill primitives. These skill primitives are implemented for each hardware platform (e.g. each KUKA model would have its own hardware-specific implementations of the individual skill primitives, etc.) When the system is connected to a specific robot, the robot-specific skill primitive implementations are used by the sequence diagram code. Using this approach we can use planning to generate these sequence diagrams for any given task, and run them on any given platform.

One important point to make here is that using a PDDL planner in some degree acts as its own verification and validation method. Assuming that the domain is specified correctly, the planner should not be able to generate plans that are infeasible given the constraints of the system. One could imagine however using PDDL to generate sequence diagrams for complex task objectives, and making any small modifications that might be required based on updates to the manufacturing process by hand. This alleviates the need to rerun complex planning problems for small adjustments to the plan. In these cases using the SysML-based V&V methods are quite useful to ensure that the updated plan still meets the constraints of the system. In cases where other high-level interaction methods are being used to create the sequence of actions (such as Programming by Demonstration), where logical-based planning methods are not employed, the SysML verification and validation tools are essential to check for successful execution and system safety.

## 6.2 Quantitative Analysis

We can use the integrated framework presented in the previous section to solve a number different problems, including those presented in the previous chapters of this dissertation such as skill abstraction and high-level interaction for robot programming. It is instructive to look at how effective this integrated GTax framework is at solving problems, though. Using the framework raises the question of whether it imparts any real benefit in these types of situations (in manufacturing scenarios), or whether it is simply another equally time-consuming alternative to the already existing methods for interacting with and programming robots.

In order to address this question effectively, we propose examining a similar situation as those posed in the skill abstraction chapter (chapter 4). In this proposed scenario we are using the setup for the model airplane assembly task. The system is given the same parts used to assemble the model airplane, and is tasked with the assembly of any number of different configurations using these pieces (see figure 32).

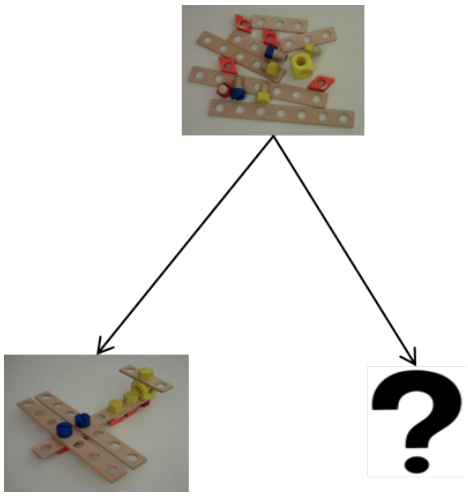


Figure 32: Assembling different model configurations using the parts from the model airplane assembly task.

In this scenario we look at what it would take for both robots to perform the assembly task. In this respect the situation is similar to the examples presented in the previous chapter on skill abstraction, except that it looks at both task and

platform problems simultaneously.

The question being addressed here again is whether in this situation using GTax is beneficial in any way compared to traditional methods for programming the robot. Because terms like *beneficial* and *better* are difficult to measure, the metric that we will use here as a comparison is lines of code used to fully implement the task on the different hardware platforms. This is a reasonable metric because lines of code can generally be correlated to time taken for implementation of the task, as well as complexity of the implementation itself. This also is an indication of code reuse, or the amount of existing code (or knowledge) being reused between successive tasks.

Figure 33 shows a graph of data gathered from this comparison. Along the x axis are individual variations on the assembly task (with 18 different configurations shown in the graph), while the y axis records the cumulative total lines of code used to run each of these assembly configurations on the two different robots (in this case the KUKA KR5 sixx and the Universal Robots UR5).

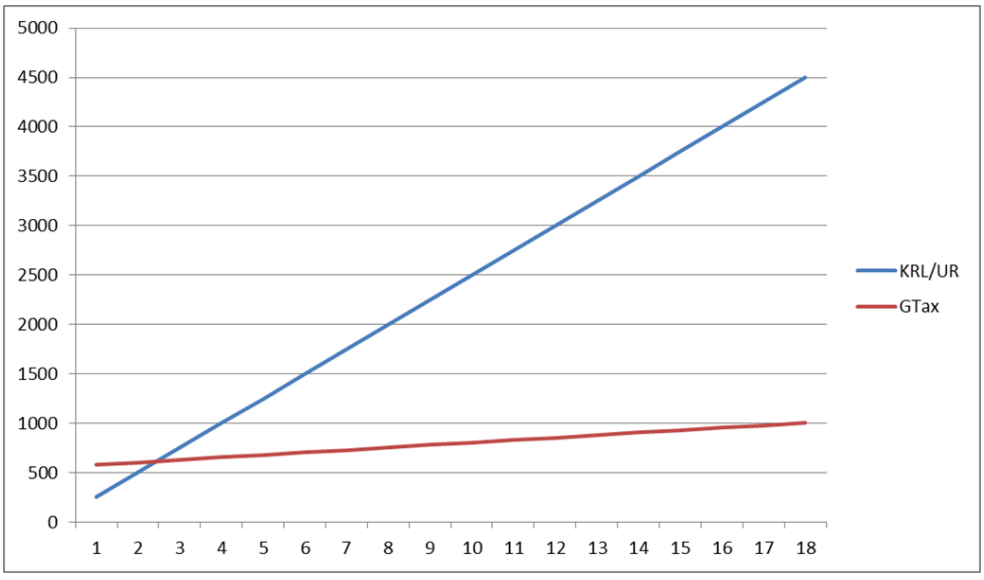



Figure 33: A Quantitative Analysis Comparing GTax to Robot Programming by Hand.

There are two interesting points to notice in the graph data. First, for low numbers of final configuration variations, hand coding requires fewer lines of code and thus

appears to be a simpler option. From here, as the number of variations increases, the required lines of code also increases for both methods. However, the required increase in lines of code for successive variations occurs at a much slower rate than the traditional hand coding method.

The reasons for this trend are worth discussing. Using GTax as a solution for this problem requires more initial lines of code than hand coding the robot motions. These lines of code come in the form of defining the PDDL domain and problem files (which include both the initial state and goal state descriptions), as well as defining the skill primitive implementations for each of the hardware platforms desired for the assembly operation. As the number of variations increases however, much of this code can be reused. The PDDL domain and initial state descriptions remain the same for each successive configuration, and the skill primitive implementations do not change. In fact, the only coding that needs to be done for each variation is the goal state description for each individual configuration. All other code (or knowledge about the system - skills, states, etc.) can be reused as is. On the other hand, designing these variations by hand for each robot controller requires the programmer to write each goal configuration from scratch for each individual configuration. Thus for any more than just a few variations, the required lines of code using the standard hand coding method increases over the GTax method considerably.

It should also be mentioned that while lines of code can give a general indication as to time required for implementation, hand coding for each robot controller also requires the programmer to specify each pose by hand by driving the robot to the desired position, which in itself is a time consuming process. As such, the trend seen in figure 33 can be seen as a best case trend for implementation time, while the actual difference between implementation time with the two approaches could be much greater.

# CHAPTER VII

# CONCLUSION & FUTURE WORK

## *7.1 Introduction*

Robotics has tremendous potential. For decades people of all walks of life have envisioned the multitude of different ways robots could make a positive benefit to society - from domestic service robots to help with household-oriented tasks, to manufacturing robots that work collaboratively with human workers to complete complex industrial objectives.

Technology today has progressed to the point that we are beginning to see this potential realized to varying degrees. Platforms such as Baxter are an important step forward, toward a true shared effort in between human laborers and robots to achieve complex manufacturing objectives, which has long been a holy grail of robotics in industrial manufacturing.

Yet despite the many advancements in the state of the art, programming robots to be robust across varied environments and objectives, and in a way that is accessible and intuitive to most users, is still a difficult task, and there remain many unmet needs with the existing technology.

This is especially true in manufacturing, where objective requirements and manufacturing processes often change. This in turn requires a change in the system configuration and programming, which is time consuming and costly. What is needed is a flexible solution that would allow robot users to easily reprogram and redeploy these robots, irrespective of changes made in the manufacturing process, or even the robot platform performing that process.

## *7.2 Contributions*

In this thesis we address the problem of inflexibility in current manufacturing processes that involve robotics and robot programming, and propose a solution in which a SysML-based task modeling framework is used to model the robot capabilities in a given application domain. By using this framework to model robot skills at a higher level of abstraction, we show that we can introduce more flexibility into the manufacturing process by addressing robot programming and interaction at the task level.

By approaching the problem in this way, we have been able to abstract away many of the difficulties with the current interaction methods, and at the same time create a framework that can span the interaction space, and thereby enable knowledge transfer in our target domain.

At the beginning of the thesis we proposed two novel contributions to the field of robotics through knowledge transfer, specifically the creation of a knowledge transfer framework that was able to span the interaction space, and a method for generating objective execution plans from user defined high-level task descriptions. In this document we have detailed the framework that has been developed, including its base language (SysML), its composition, and its use and application. We have demonstrated how through appropriate abstractions we were able to span the interaction space along the vertical dimension, allowing us to use a high-level interaction method such as PDDL to specify a task plan, and using the abstractions represented in the framework create a plan that could be run on a real robot system. We demonstrate how we could span the interaction space along the horizontal dimension by demonstrating how the framework could be used to generalize descriptions across different applications, through task abstraction and platform abstraction. These experiments have shown that the knowledge transfer framework is able to span the interaction space, and that we have at least one high-level interaction method for generating

task plans automatically based on user descriptions.

## *7.3 Discussion*

While we have been able to demonstrate some of the benefits of utilizing this abstraction framework for knowledge transfer, there are some questions that do arise. One such question pertains to how well this manufacturing assembly domain model would work given significant changes to parameters for a given problem. For example, suppose a robot was tasked with assembling the model airplane, only instead of being given the Baufix wooden construction parts, the system was required to use Lego building blocks instead. How well would the knowledge transfer between the tasks in this case? This example can be used to illustrate what kinds of knowledge transfer are possible with the GTax framework. Using the integrated approach presented in the previous chapter, we can look at each step to see what type of knowledge transfers in this case, and what does not. As discussed in the chapter on skill abstraction, not every skill primitive modeled in the manufacturing assembly taxonomy is required for performing specific assembly tasks. This would be true in the Lego case as well. Figure 34 shows the subset of the taxonomy that would be needed for performing assembly tasks using the Lego parts. The sections of the taxonomy that are shaded gray are not needed for performing assembly tasks using the Lego pieces. For the planning phase, while the end model may look the same, the plans for achieving those models would in fact be quite different. Each model requires different parts, and a different number and configuration of those parts. Thus while the domain description itself remains the same, the initial state of the system would change (given different components used for manipulation and assembly), as well as the goal state and the plan itself.

Given that the action plan generated by the PDDL planner is different in each of these cases, it follows that the sequence diagrams would be different as well. The

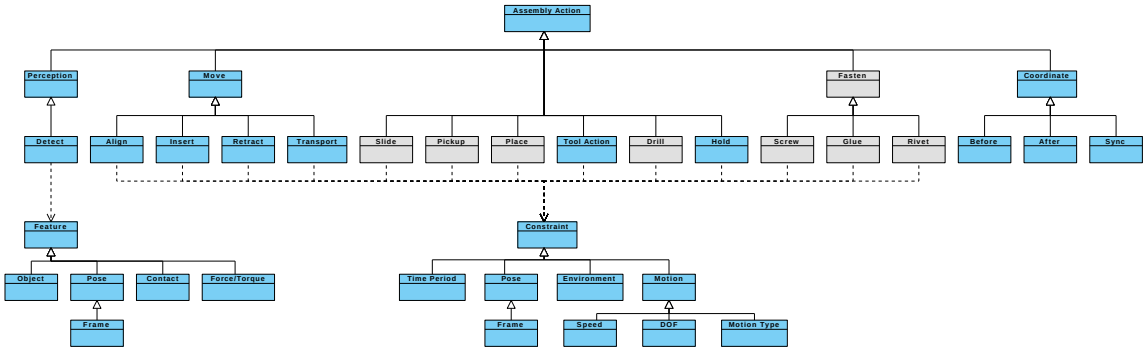



Figure 34: Taxonomy subset for Lego assembly.

parameters required to specify the skill primitives would be specific to the different parts used to build the model. It should be noted however that the skill primitives composing the sequence diagrams will remain the same, as the domain itself has not changed. (In fact given that the Legos are simpler construction parts that in general are meant only for stacking, the number of different skill primitives utilized for the Lego plan should be fewer.) The parts are still picked up and placed in different locations for the actual assembly of the model. One assumption is that the system is familiar with the Lego parts themselves and how to manipulate them, or that it has some ability to learn these interaction parameters. From here, the systems would be identical. As the skill primitives have not changed, the robot-specific implementations would also remain the same. The only difference would be that the robots are running different sequence diagrams to achieve the same result - assembly of the model airplane.

Another important question to address is what can be said about the quality of the taxonomic model itself. How do we know that the taxonomy presented is the *right* taxonomy for modeling the given domain, and what is the extent of what can be modeled? The taxonomy presented in this work was formulated and refined based on the analysis of several different manufacturing assembly tasks. A taxonomy is the right taxonomy for modeling an application domain if it can model the tasks within

that domain. The taxonomy utilized here can be used to model any task that can be decomposed into a sequence of actions or skills that have been modeled within taxonomy itself. For example, based on the ability of the framework to successfully model the Cranfield Benchmark, we know that we can model any tasks that can be decomposed into a sequence of pick, place, and insert operations. While this is certainly not all assembly tasks, it does cover many. We can also model tasks that require the operation of specialized tooling.

However, in its current formulation, we cannot make any claims about the completeness of the taxonomy for modeling any given task in the application domain. What can be said in this regard is whether a given task can be modeled. The PDDL planning approach can be used as a completeness checker, or a method for verifying whether a specific task can be successfully modeled using the taxonomy. Given an accurate definition of the application domain in the PDDL domain file, if a plan can be found for achieving a task goal state based on the domain and initial state of the problem, then there exists a sequence diagram using the skill primitives in the taxonomy that will achieve that task objective.

## *7.4 Future Work*

While this work has been able to demonstrate a method for making the manufacturing process more flexible in the application domain, still there remain open questions and opportunities for future work.

One assumption that was utilized during the work undertaken in this thesis is that of perfect execution. That is to say that the expectation was that when a robot was tasked with performing a specific skill, it would always be able to successfully accomplish that task. This was done for the sake of simplicity in demonstrating the strength of the framework itself, while ignoring the questions of fault detection and fault recovery. This area, fault recovery (also commonly known as error recovery or

exception handling) is one that would greatly benefit the applicability of the GTax framework. There are a number of reasons that a skill being executed would not succeed in the real world, from imperfect pose estimation and errors in modeling specific things in the environment, to a whole host of other unknown and unexpected issues. Having the ability to first detect, diagnose, and then recover from execution failures would increase the robustness of the framework. Note that the ability to detect features in the environment that would indicate failure already exists in the taxonomy in the ***Detect*** skill primitive.

Another assumption used in the experiments in this work was the feasibility of all of the actions and knowledge transfer used in the system. Given different tasks, it was assumed that the platform was able to perform all actions necessary to complete the tasks (all parts were reachable, all poses achievable, etc.) Given different platforms, it was assumed that all platforms possessed the capabilities needed to successfully perform required actions (e.g. possessed the necessary degrees of freedom, etc.) This is not an unreasonable assumption given the application domain. Manufacturing environments are often highly structured and highly engineered, where for example reachability analysis is performed prior to execution, and the environment is structured so that this is true for given tasks. That said, it would improve the utility of this knowledge transfer framework if this ability was integrated, the ability to check desired task plans against platform constraints (workspace, tooling, etc.)

This work has demonstrated how the knowledge transfer framework can be used to enable knowledge transfer in the manufacturing assembly domain. This begs the question of how effective it would be at enabling knowledge transfer in other application domains. Future work in this area could address the issue of how well this approach would transfer to different application domains, such as laboratory automation. There are a number of interesting questions that would need to be addressed, such as what knowledge could be transferred (indicating general robot capabilities)

and what knowledge could not easily be transferred (possibly indicating domain-specific knowledge.) It would also be important to identify domains where perhaps this type of knowledge transfer framework might not be appropriate. These kinds of domain could include highly dynamic environments where there is typically very little repetition of tasks or reuse of common skills.

# REFERENCES


[1] Abbas, T. and MacDonald, B., "Generalizing topological task graphs from multiple symbolic demonstrations in programming by demonstration (pbd) processes," in *Robotics and Automation (ICRA), 2011 IEEE International Conference on*, pp. 3816 –3821, may 2011.

[2] Balakirsky, S., Kootbally, Z., Schlenoff, C., Kramer, T., and Gupta, S., "An industrial robotic knowledge representation for kit building applications," in *IEEE/RSJ International Conference on Intelligent Robots and Systems*, (San Francisco), 2012.

[3] Benton, J., Coles, A., and Coles, A., "Temporal planning with preferences and time-dependent continuous costs," 2012.

[4] Boehm, B., "Verifying and validating software requirements and design specifications," *Software, IEEE*, vol. 1, pp. 75–88, Jan 1984.

[5] Bradski, G., "The OpenCV Library," *Dr. Dobb's Journal of Software Tools*, 2000.

[6] Cakmak, M., *Guided teaching interactions with robots: embodied queries and teaching heuristics*. PhD thesis, Georgia Institute of Technology, Atlanta, GA, USA, April 2012.

[7] Calinon, S., Guenter, F., and Billard, A., "On learning, representing, and generalizing a task in a humanoid robot," *Systems, Man, and Cybernetics, Part B: Cybernetics, IEEE Transactions on*, vol. 37, pp. 286 –298, april 2007.

[8] Carneiro, E., Maciel, P., Callou, G., Tavares, E., and Nogueira, B., "Mapping sysml state machine diagram to time petri net for analysis and verification of embedded real-time systems with energy constraints," in *Advances in Electronics and Micro-electronics, 2008. ENICS '08. International Conference on*, pp. 1–6, Sept 2008.

[9] Chao, C., Lee, J., Begum, M., and Thomaz, A. L., "Simon plays simon says: The timing of turn-taking in an imitation game," in *RO-MAN, 2011 IEEE*, pp. 235–240, IEEE, 2011.

[10] Chhaniyara, S., Saaj, C., Meadiger, B., Althoff-Kotzias, M., and Ahrns, I., "SysML based system engineering: A case study for space robotic systems," in *62nd Intl. Astronautical Congress, Cape Town*, 2011.

[11] Collins, K., Palmer, A., and Rathmill, K., "The development of a European benchmark for the comparison of assembly robot programming systems," in *Proceedings of the 1st Robotics Europe Conference, Brussels*, pp. 27–28, 1984.

[12] Dantam, N. and Stilman, M., "The motion grammar: Linguistic perception, planning, and control," in *Robotics: Science and Systems*, 2011.

[13] Dillman, R., Rogalla, O., Ehrenman, M., Zoellner, R., and Bordegoni, M., "Learning robot behavior and skills based on human demonstration and advice: The machine learning paradigm," in *Proceedings of the 9th International Symposium of Robotics Research ISRR99*, (Snowbird, USA), pp. 181–190, October 1999.

[14] Dillmann, R., "Teaching and learning of robot tasks via observation of human performance," *Robotics and Autonomous Systems*, vol. 47, no. 2, pp. 109–116, 2004.

[15] Ekvall, S., Aarno, D., and Kragic, D., "Task learning using graphical programming and human demonstrations," in *Robot and Human Interactive Communication, 2006. ROMAN 2006. The 15th IEEE International Symposium on*, pp. 398 –403, sept. 2006.

[16] Ekvall, S. and Kragic, D., "Learning task models from multiple human demonstrations," in *Robot and Human Interactive Communication, 2006. ROMAN 2006. The 15th IEEE International Symposium on*, pp. 358 –363, sept. 2006.

[17] Fikes, R. E. and Nilsson, N. J., "STRIPS: A new approach to the application of theorem proving to problem solving," *Artificial Intelligence*, vol. 2, pp. 189–208, 1971.

[18] Finucane, C., Jing, G., and Kress-Gazit, H., "Ltlmop: Experimenting with language, temporal logic and robot control," in *Intelligent Robots and Systems (IROS), 2010 IEEE/RSJ International Conference on*, pp. 1988 –1993, oct. 2010.

[19] Forge, S. and Blackman, C., "A helping hand for europe: the competitive outlook for the eu robotics industry," tech. rep., Institute for Prospective Technological Studies, Joint Research Center, European Commission, 2010.

[20] Friedenthal, S., Moore, A., and Steiner, R., *A Practical Guide to SysML: Systems Modeling Language*. The MK/OMG Press, Elsevier Science, 2008.

[21] Friedrich, H., Kaiser, M., and Dillmann, R., "What can robots learn from humans?," in *Annual Reviews in Control* (Gertler, J., ed.), vol. 20, ch. Robotics, pp. 167–172, Elsevier, 1997.

[22] Gat, E., "Esl: a language for supporting robust plan execution in embedded autonomous agents," in *Aerospace Conference, 1997. Proceedings., IEEE*, vol. 1, pp. 319 –324 vol.1, feb 1997.

[23] Gerevini, A. and Long, D., "Preferences and soft constraints in PDDL3," *Proceedings of ICAPS workshop on Planning with Preferences and Soft Constraints*, pp. 46–53, 2006.

[24] Hause, M., "The sysml modelling language," in *Fifteenth European Systems Engineering Conference*, Citeseer, 2006.

[25] Hersch, M., Guenter, F., Calinon, S., and Billard, A., "Dynamical system modulation for robot learning via kinesthetic demonstrations," *Robotics, IEEE Transactions on*, vol. 24, no. 6, pp. 1463–1467, 2008.

[26] Hitzler, P., Krötzsch, M., Parsia, B., Patel-Schneider, P. F., and Rudolph, S., eds., *OWL 2 Web Ontology Language: Primer*. W3C Recommendation, 27 October 2009. Available at `http://www.w3.org/TR/owl2-primer/`.

[27] Homem de Mello, L. and Sanderson, A., "And/or graph representation of assembly plans," *Robotics and Automation, IEEE Transactions on*, vol. 6, no. 2, pp. 188–199, Apr.

[28] Huckaby, J. and Christensen, H. I., "A taxonomic framework for task modeling and knowledge transfer in manufacturing robotics," in *8th International Cognitive Robotics Workshop*, July 2012.

[29] Huckaby, J., Vassos, S., and Christensen, H. I., "Planning with a task modeling framework in manufacturing robotics," in *IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2013.

[30] Ijspeert, A., Nakanishi, J., Shibata, T., and Schaal, S., "Nonlinear dynamical systems for imitation with humanoid robots," in *Proceedings of the IEEE International Conference on Humanoid Robots*, pp. 219–226, 2001.

[31] Jarraya, Y. and Debbabi, M., "Formal specification and probabilistic verification of sysml activity diagrams," in *Theoretical Aspects of Software Engineering (TASE), 2012 Sixth International Symposium on*, pp. 17–24, July 2012.

[32] Kaelbling, L. P. and Lozano-Pérez, T., "Hierarchical planning in the now," in *IEEE Conference on Robotics and Automation (ICRA)*, 2011. Finalist, Best Manipulation Paper Award.

[33] Kaiser, M. and Dillmann, R., "Robot programming based on single demonstration and user intentions," in *3rd European Workshop on Learning Robots*, (Heraklion, Greece), April 1995.

[34] Konidaris, G., Kuindersma, S., Grupen, R., and Barto, A., "Robot learning from demonstration by constructing skill trees," *The International Journal of Robotics Research*, 2011.

[35] Kosecka, J. and Bajcsy, R., "Discrete event systems for autonomous mobile agents," *Robotics and Autonomous Systems*, vol. 12, pp. 187–198, 1993.

[36] Kress-Gazit, H., Wongpiromsarn, T., and Topcu, U., "Correct, reactive, high-level robot control," *Robotics Automation Magazine, IEEE*, vol. 18, pp. 65 –74, sept. 2011.

[37] Krüger, N., Geib, C., Piater, J., Petrick, R., Steedman, M., Wörgötter, F., Ude, A., Asfour, T., Kraft, D., Omrčen, D., Agostini, A., and Dillmann, R., "Object-Action Complexes: Grounded Abstractions of Sensory-motor Processes," *Robotics & Autonomous Systems*, vol. 59, pp. 740–757, 10 2011.

[38] Kuniyoshi, Y., Inaba, M., and Inoue, H., "Learning by watching: extracting reusable task knowledge from visual observation of human performance," *Robotics and Automation, IEEE Transactions on*, vol. 10, pp. 799 –822, dec 1994.

[39] Lenat, D. B., "Cyc: A large-scale investment in knowledge infrastructure," *Communications of the ACM*, vol. 38, no. 11, pp. 33–38, 1995.

[40] Lim, G. H., Suh, I. H., and Suh, H., "Ontology-based unified robot knowledge for service robots in indoor environments," *Systems, Man and Cybernetics, Part A: Systems and Humans, IEEE Transactions on*, vol. 41, pp. 492 –509, may 2011.

[41] Liu, H. and Singh, P., "Conceptnet: A practical commonsense reasoning toolkit," *BT Technology Journal*, vol. 22, pp. 211–226, 2004.

[42] Lyons, D. and Arbib, M., "A formal model of computation for sensory-based robotics," *Robotics and Automation, IEEE Transactions on*, vol. 5, pp. 280 –293, jun 1989.

[43] Mcdermott, D., Ghallab, M., Howe, A., Knoblock, C., Ram, A., Veloso, M., Weld, D., and Wilkins, D., "PDDL - The Planning Domain Definition Language," tech. rep., CVC TR-98-003, Yale Center for Computational Vision and Control, 1998.

[44] Nicolescu, M. N. and Mataric, M. J., "Natural methods for robot task learning: Instructive demonstrations, generalization and practice," in *In Proceedings of the Second International Joint Conference on Autonomous Agents and Multi-Agent Systems*, pp. 241–248, 2003.

[45] Nilsson, K. and Johansson, R., "Integrated architecture for industrial robot programming and control," *Robotics and Autonomous Systems*, vol. 29, pp. 205–226, 1999.

[46] OMG, "OMG Systems Modeling Language (OMG SysML) Version 1.2 Specification," 2010.

[47] Pardowitz, M., Knoop, S., Dillmann, R., and Zollner, R., "Incremental learning of tasks from user demonstrations, past experiences, and vocal comments," *Systems, Man, and Cybernetics, Part B: Cybernetics, IEEE Transactions on*, vol. 37, pp. 322 –332, april 2007.

[48] Pednault, E. P. D., "ADL: Exploring the middle ground between STRIPS and the situation calculus.," in *Proceedings of the International Conference on Principles of Knowledge Representation and Reasoning (KR)*, pp. 324–332, 1989.

[49] Peters, J. R., *Machine learning of motor skills for robotics*. PhD thesis, University of Southern California, Los Angeles, CA, USA, April 2007. AAI3262746.

[50] Pin, F. G., Parker, L. E., and DePiero, F. W., "On the design and development of a human-robot synergistic system," *Robotics and Autonomous Systems*, vol. 10, no. 2–3, pp. 161 – 184, 1992.

[51] Quigley, M., Conley, K., Gerkey, B., Faust, J., Foote, T. B., Leibs, J., Wheeler, R., and Ng, A. Y., "ROS: an open-source robot operating system," in *International Conference on Robotics and Automation*, Open-Source Software workshop, May 2009.

[52] Rahman, M. A. A., Mayama, K., Takasu, T., Yasuda, A., and Mizukawa, M., "Model-driven development of intelligent mobile robot using systems modeling language (SysML)," in *Mobile Robots - Control Architectures, Bio-Interfacing, Navigation, Multi Robot Motion Planning and Operator Training*, InTech, 2011.

[53] Rethink Robotics, "Baxter." `http://www.rethinkrobotics.com/index.php/products/baxter`. Accessed 1 April 2013.

[54] Rogalla, O. and Ehrenmann, M., "A sensor fusion approach for PbD," in *Proceedings of the International Conference on Intelligent Robots and Systems (IROS)*, 1998.

[55] Rosell, J., "Assembly and task planning using petri nets: a survey," *Proceedings of the Institution of Mechanical Engineers, Part B: Journal of Engineering Manufacture*, vol. 218, no. 8, pp. 987–994, 2004.

[56] ROSETTA, "The ROSETTA project page.." `http://fp7rosetta.org`, 2009. Accessed 10 March 2012.

[57] Rowe, S. and Wagner, C. R., "An introduction to the joint architecture for unmanned systems (jaus)," *Ann Arbor*, vol. 1001, p. 48108.

[58] Rusu, R. B. and Cousins, S., "3D is here: Point Cloud Library (PCL)," in *IEEE International Conference on Robotics and Automation (ICRA)*, (Shanghai, China), May 9-13 2011.

[59] Santos, W. A., Leonor, B. B. F., and Stephany, S., "A knowledge-based and model-driven requirements engineering approach to conceptual satellite design," in *Conceptual Modeling - ER 2009* (Laender, A. H. F., Castano, S., Dayal, U., Casati, F., and Oliveira, J. P. M., eds.), vol. 5829 of *Lecture Notes in Computer Science*, pp. 487–500, Springer Berlin Heidelberg, 2009.

[60] SIARAS, "Final Report - SIARAS (Skill-based Inspection and Assembly for Reconfigurable Automation Systems)." http://cordis.europa.eu/search/index.cfm?fuseaction=lib.document&DOC_LANG_ID=EN&DOC_ID=121979181&q=, 2011.

[61] Simmons, R. and Apfelbaum, D., "A task description language for robot control," in *Proceedings Conference on Intelligent Robotics and Systems*, October 1998.

[62] Singh, P., Lin, T., Mueller, E. T., Lim, G., Perkins, T., and Zhu, W. L., "Open mind common sense: Knowledge acquisition from the general public," in *On the Move to Meaningful Internet Systems 2002: CoopIS, DOA, and ODBASE*, pp. 1223–1237, Springer, 2002.

[63] Skubic, M. and Volz, R., "Acquiring robust, force-based assembly skills from human demonstration," *Robotics and Automation, IEEE Transactions on*, vol. 16, pp. 772 –781, dec 2000.

[64] Soetens, P. and Bruyninckx, H., "Realtime hybrid task-based control for robots and machine tools," in *IEEE International Conference on Robotics and Automation*, pp. 260–265, 2005.

[65] Tenorth, M. and Beetz, M., "KnowRob — Knowledge Processing for Autonomous Personal Robots," in *IEEE/RSJ International Conference on Intelligent RObots and Systems.*, pp. 4261–4266, 2009.

[66] Tenorth, M., Nyga, D., and Beetz, M., "Understanding and Executing Instructions for Everyday Manipulation Tasks from the World Wide Web.," in *IEEE International Conference on Robotics and Automation (ICRA).*, pp. 1486–1491, 2010.

[67] Ude, A., "Trajectory generation from noisy positions of object features for teaching robot paths," *Robotics and Autonomous Systems*, vol. 11, no. 2, pp. 113–127, 1993.

[68] Waibel, M., Beetz, M., D'Andrea, R., Janssen, R., Tenorth, M., Civera, J., Elfring, J., Gálvez-López, D., Häussermann, K., Montiel, J., Perzylo, A., Schießle, B., Zweigle, O., and van de Molengraft, R., "RoboEarth - A World Wide Web for Robots," *Robotics & Automation Magazine*, vol. 18, no. 2, 2011. Accepted for publication.

[69] Zweigle, O., van de Molengraft, R., d'Andrea, R., and Häussermann, K., "Roboearth: connecting robots worldwide," in *Proceedings of the 2nd International Conference on Interaction Sciences: Information Technology, Culture and Human*, ICIS '09, (New York, NY, USA), pp. 184–191, ACM, 2009.